



# **PARTIE 2 : CONCEPTION**

## **Chapitre 5 :**

### **Conception statique**

**Compléments sur le diagramme de classes :  
Diagramme de classes de conception**

**Diagramme de composants**

**Diagramme de déploiement**

3<sup>ème</sup> année A

Année universitaire :  
2020-2021

Henda SFAXI  
[henda.sfaxi@esprit.tn](mailto:henda.sfaxi@esprit.tn)



# Plan

- Compléments sur le diagramme de classes : Diagramme de classes de conception
  - Concepts avancés:
    - [Attributs dérivés](#)
    - [Classe-association](#)
    - [Navigabilité d'une association](#)
    - [Liens de dépendance](#)
  - Diagramme d'interaction – Diagramme de classes:
    - Liens entre le diagramme d'états-transition et le diagramme de classes
    - Liens entre le diagramme de séquences objet et le diagramme de classes
- [Diagramme de composants](#)
- [Diagramme de déploiement](#)



# Compléments sur le diagramme de classes : Diagramme de classes de conception

Concepts avancés

# Rappel



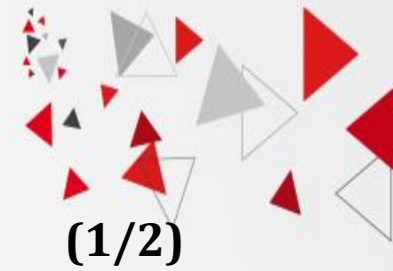
- Une application possède **1..1** diagramme de classes

	Analyse	Conception
Appellation	Diagramme de classes <i>d'analyse</i>	Diagramme de classes <i>de conception</i>
Rôle	Description préliminaire de la structure des classes candidates	Description détaillée de toutes les classes nécessaires à l'implémentation
Types de classes	Entités ou modèles (Pincipalement)	Tous les types : Gestionnaire ou contrôleurs, entités, classes interfaces, interface graphique...
Version	V1.0	VFinale



# Attribut dérivé

# ► Attribut dérivé



- Les attributs dérivés sont calculables à partir d'autres attributs et précédé par un /
- Un attribut dérivé peut être traduit
  - soit en un attribut et une opération qui met à jour et retourne la valeur de l'attribut à chaque appel
  - soit en une opération qui calcule la valeur lors de chaque appel (sans stockage de l'attribut) → *conseillé*

# ► Attribut dérivé

(2/2)

- Exemple :

## Conception

### Analyse

Abonne	
- numInscri	: String
- cin	: int
- dateNaiss	: Date
/age	: int



ou



Abonne	
- numInscri	: String
- cin	: int
- dateNaiss	: Date
/age	: int
+ calculAge()	: int

Abonne	
- numInscri	: String
- cin	: int
- dateNaiss	: Date
+ calculAge()	: int



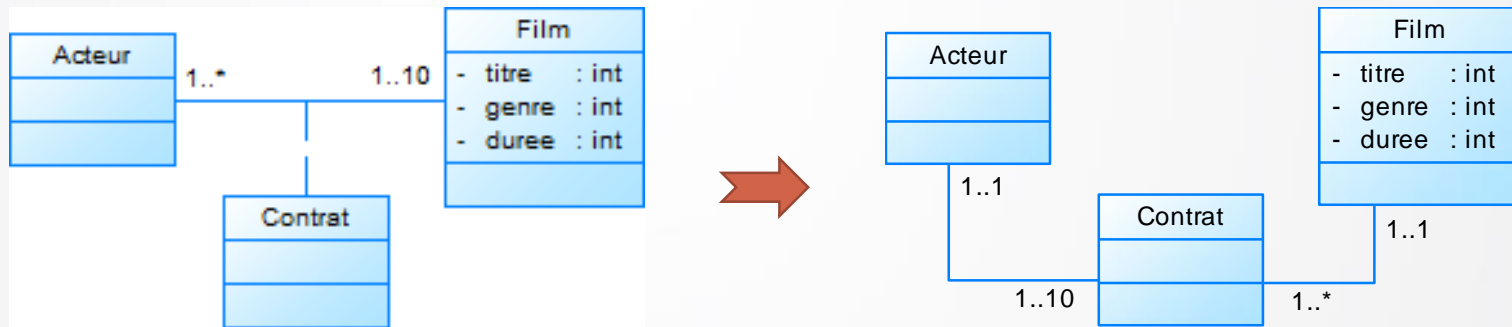
# Classe-association



# ▶ Classe-association

(1/2)

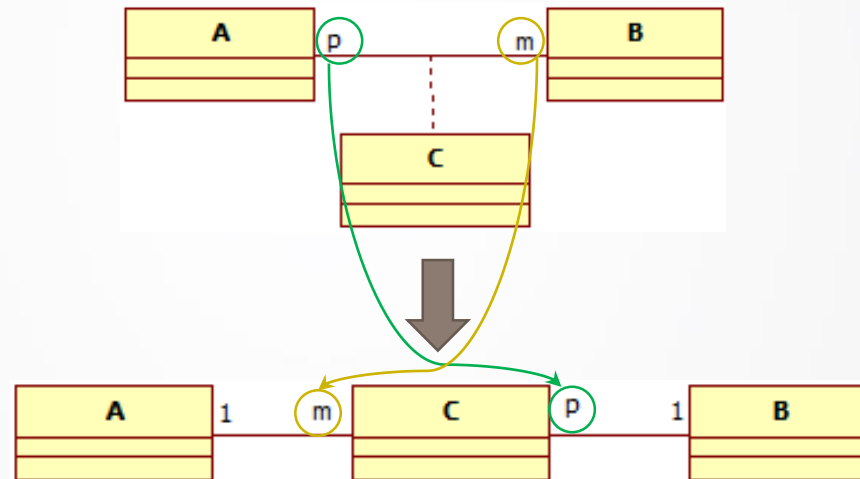
- La transformation la plus courante en conception:
  - La classe d'association devient une classe intermédiaire
  - L'association initiale est découpée en deux



# ► Classe-association

(2/2)

- Plus généralement :





# Navigabilité d'une association

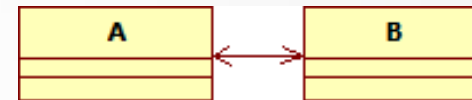
# ► Navigabilité d'une association

(1/3)

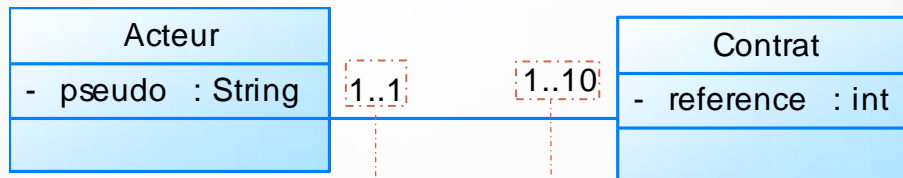
- Une association est par défaut **bidirectionnelle**:
  - **Navigable** dans les deux sens



Ou



- Conséquence :



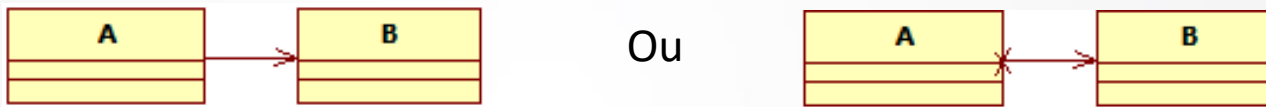
```
public class Acteur {  
    public String pseudo;  
    private List<Contrat> mesContrats;  
    ...  
}
```

```
public class Contrat {  
    public int reference;  
    private Acteur acteur;  
    ...  
}
```

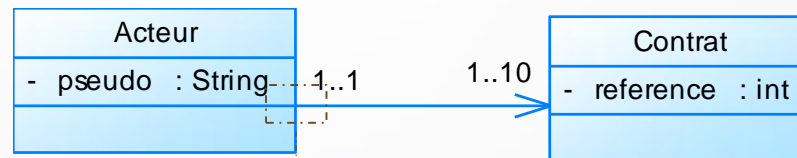
Ou tout autre objet qui implémente *Collection*

# ► Navigabilité d'une association (2/3)

- Une association peut être *unidirectionnelle*
  - *Navigable* dans un seul sens



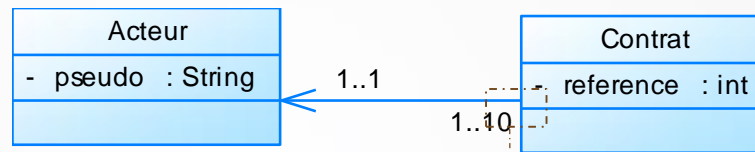
- L'association est navigable de A vers B seulement
- Conséquences :



```
public class Acteur {  
    public String pseudo;  
    private List<Contrat> mesContrats;  
    ...  
}
```

```
public class Contrat {  
    public int reference;  
    private Acteur acteur;  
    ...  
}
```

# ► Navigabilité d'une association (3/3)



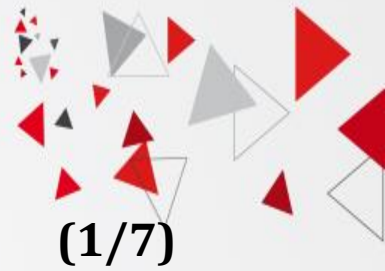
```
public class Acteur {
    public String pseudo;
    private List<Contrat> mesContrats;
    ...
}
```

```
public class Contrat {
    public int reference;
    private Acteur acteur;
    ...
}
```



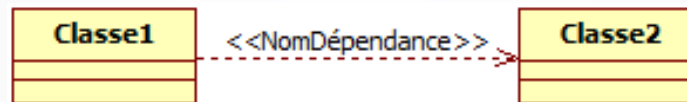
# Liens de dépendance

# ► Liens de dépendance



(1/7)

- Relation unidirectionnelle (lien en pointillé) exprimant une dépendance entre éléments :
  - La modification de la cible modifie la source
  - Lien temporaire : à l'exécution
- Représentation graphique:



- *Classe1* n'a pas de relation avec *Classe2*
- *Classe1* dépend de *Classe2*
  - S'il y a une mise à jour sur *Classe2*
  - Alors *Classe1* doit être mise à jour



# ► Liens de dépendance

(2/7)

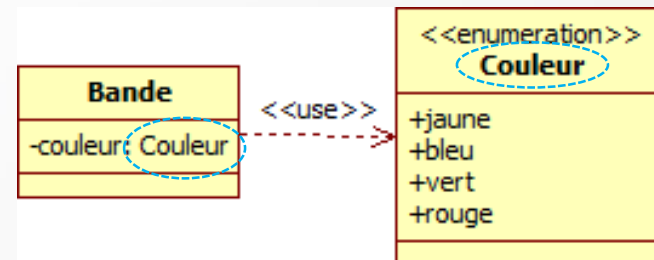
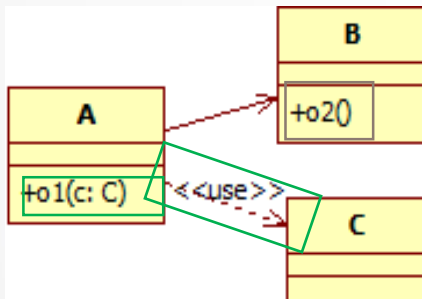
- Plusieurs types de dépendance, dont :

- « *Use* » : 

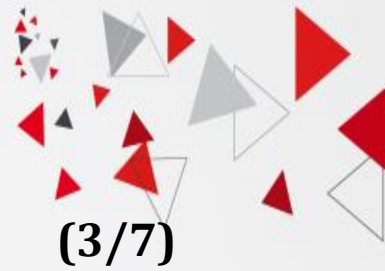
- Utilisé lorsque:

- La méthode d'une classe1 utilise comme paramètre un objet de type classe2
- La classe1 possède un attribut de type classe2
  - Exemples : Classe *Énumération* et classe *Structure*

- Exemples :

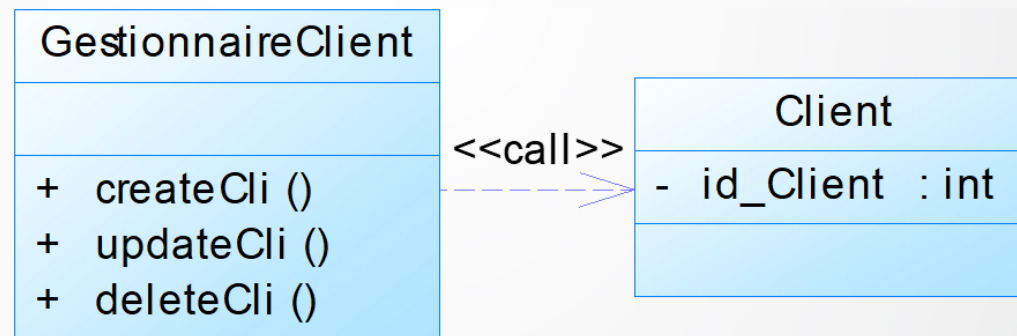


# ► Liens de dépendance

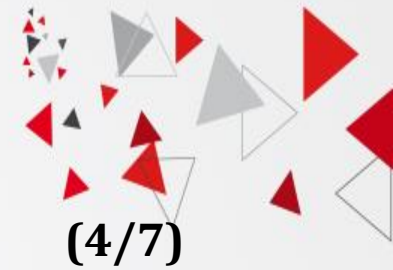


(3/7)

- « *Call* »:
  - Utilisé lorsqu’:
    - Une méthode d’une classe1 invoque une méthode d’une classe2
  - Exemple :

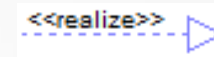


# ► Liens de dépendance



(4/7)

- « **Realize** » ou « **Implements** »:

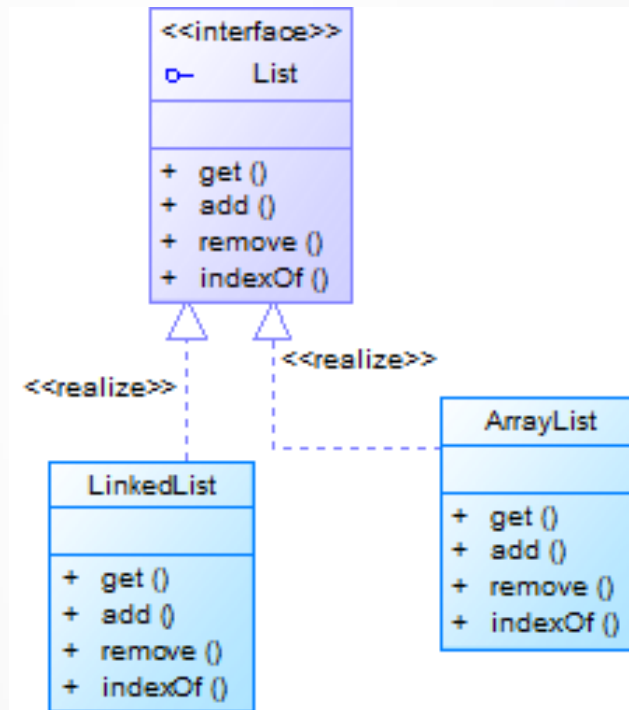


- Utilisé lorsqu':
  - Une classe *implémente* une interface
- Rappel :
  - Une interface est une classe dont toutes les méthodes sont abstraites
  - Une interface spécifie un contrat à respecter par les classes qui *réalisent* ou *implémentent* cette interface :
    - Toutes les méthodes de l'interface doivent être implémentées par ces classes

# ► Liens de dépendance

(5/7)

- Exemple 1 :

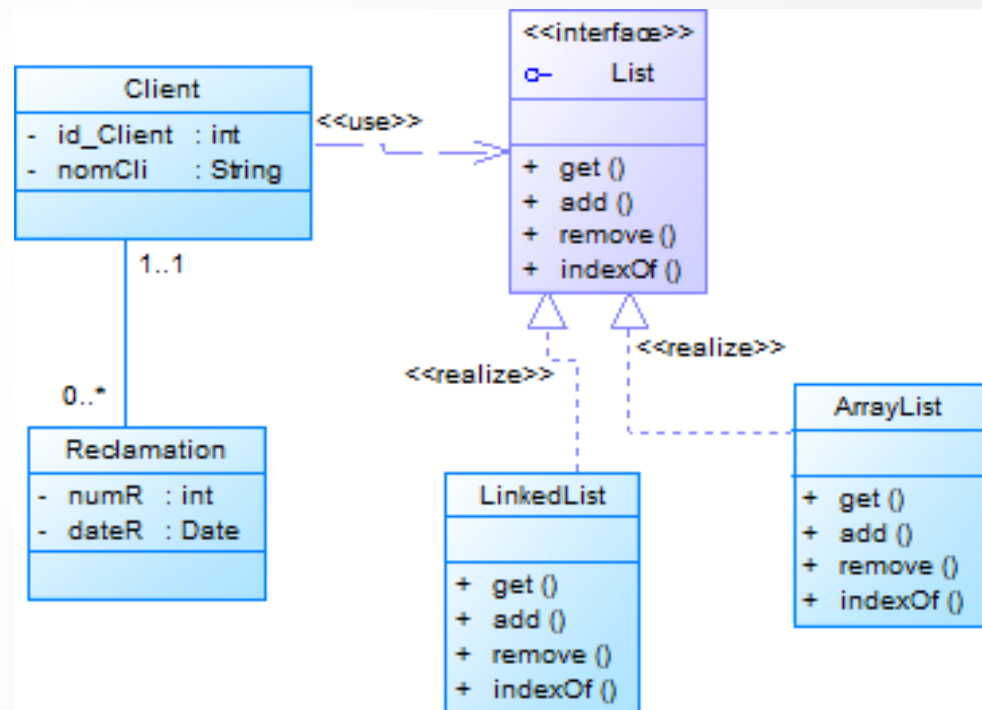


# ► Liens de dépendance

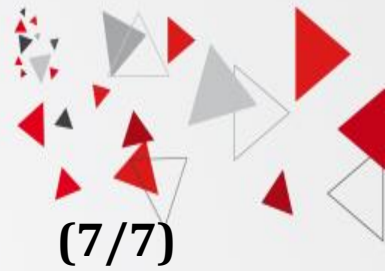
(6/7)

- Exemple 2 :

```
public class Client {  
  
    public int id_client;  
  
    private List<Reclamation>  
    reclamations;  
    ...  
  
    List<Reclamation> = new  
    ArrayList<Reclamation>;  
    --Ou  
    List<Reclamation> = new  
    LinkedList<Reclamation>;  
    ...  
}
```



# Liens de dépendance



- Remarque:
  - Les liens de dépendance sont utilisés dans les diagrammes :
    - de classes
    - de packages
    - de composants
    - de déploiement

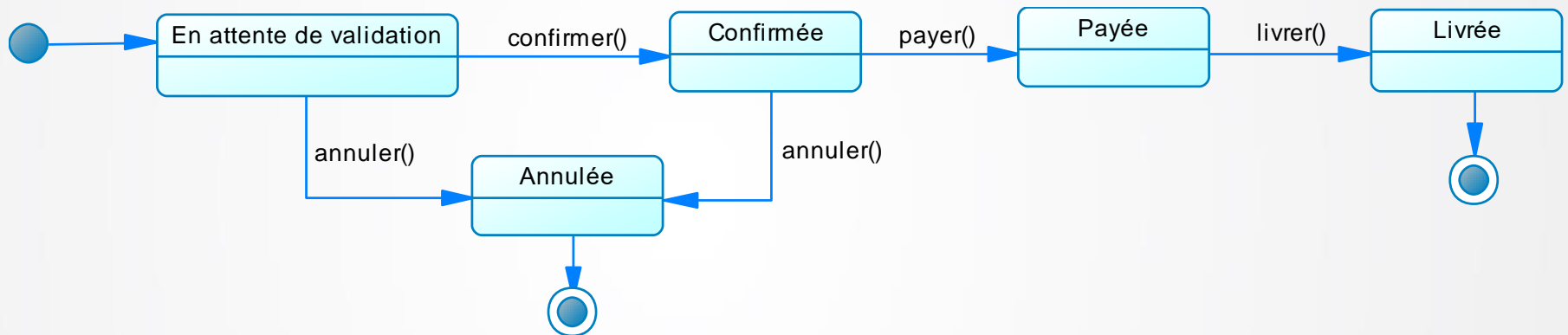


# Diagramme d'états-transition - Diagramme de classes

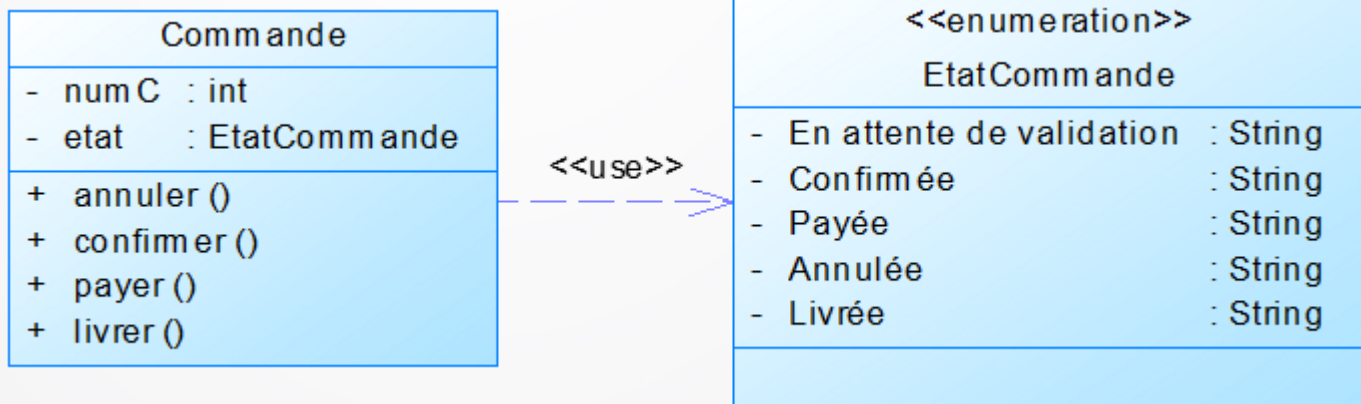
# ► D. E-T – D. C.



- Reprenons cet exemple:



- Conséquence :





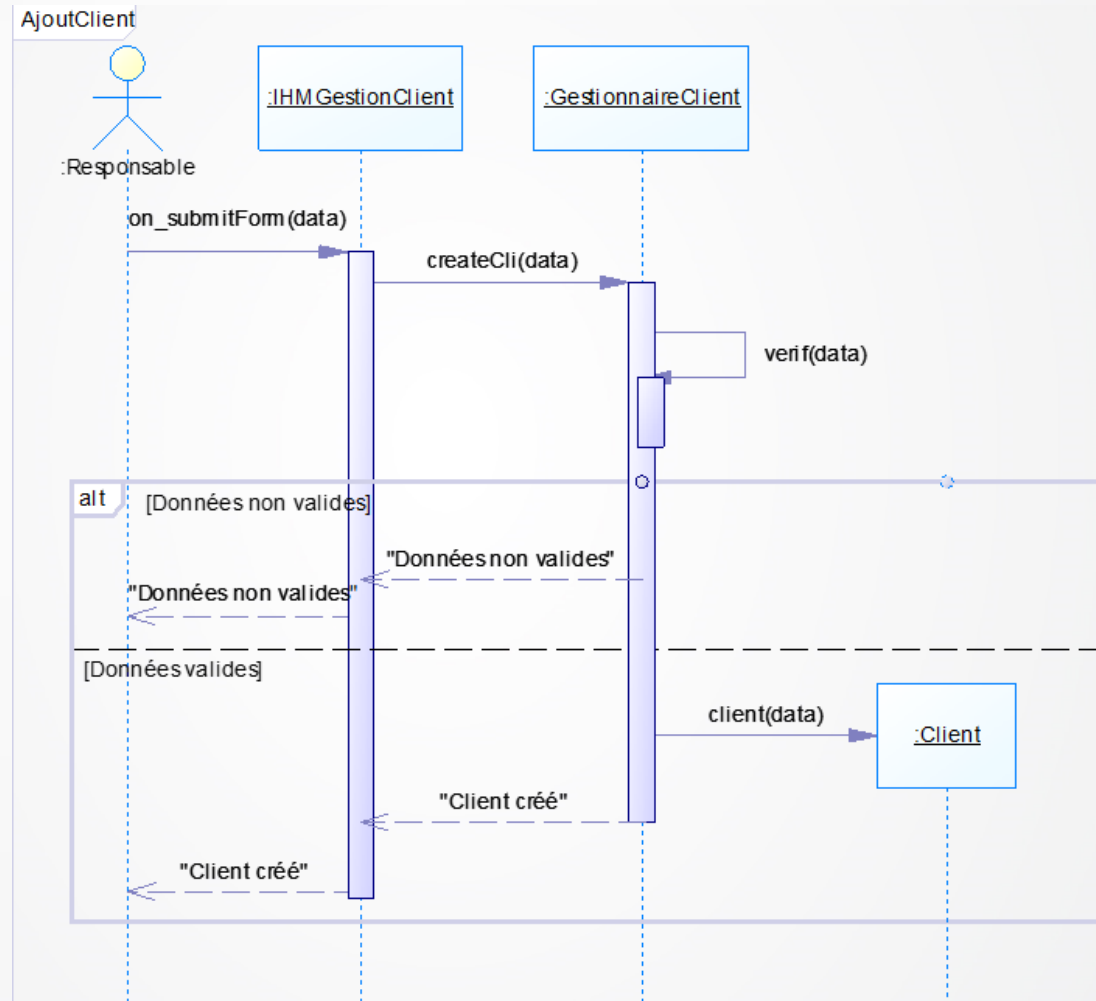


# Diagramme de séquences objet - Diagramme de classes

# ► D. S.O. – D. C.

(1/3)

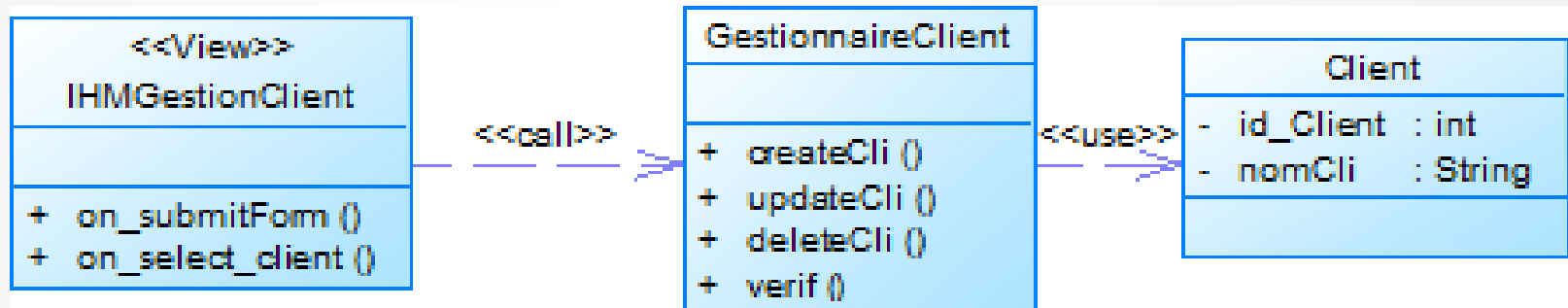
- Exemple :



# ► D. S.O. – D. C.

(2/3)

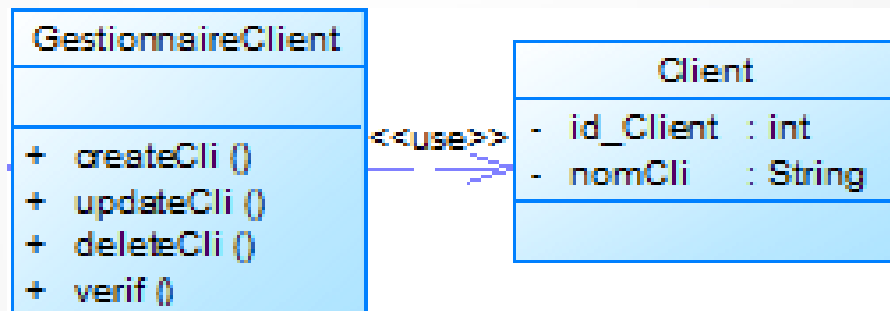
- Conséquence :



# ► D. S.O. – D. C.

(3/3)

- Remarque :
  - Lorsque les interfaces graphiques ne sont pas des classes (exemples : .js, .html, .xhtml) :
    - Elles figurent dans le diagramme de séquences objet
    - Elles ne figurent pas dans le diagramme de classes
  - Exemple :



# ▶ Remarque

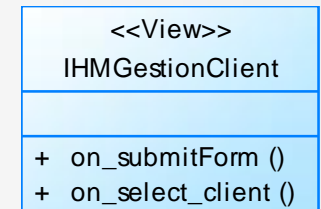


- Attention : 

Ne pas confondre IHM (View) et classe interface

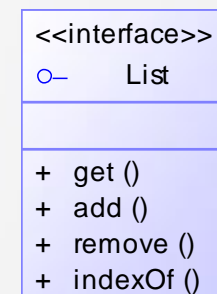
- ***IHM*** :

- Interface Homme Machine (fr)
  - Appelée aussi ***GUI*** Graphical User Interface (en)
- Interface graphique



- Classe interface :

- Classe qui ne contient que des méthodes abstraites





# Diagramme de composants

Présentation

Composant

Représentations



# Présentation



# Présentation

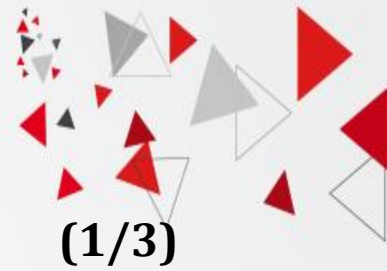
- Utilisé pour décrire le système d'un point de vue implémentation
- Offre une vue de haut niveau de l'architecture *logique* du système
- Permet :
  - de regrouper des éléments du système au sein de modules appelés composants
    - Exemples d'éléments : classes, classes interfaces, fichiers
  - de décrire les interactions entre les composants





# Composant


# Composant



- Élément **logiciel** qui **fournit un service** bien précis
  - **Réutilisable** et **remplaçable**
- **Unité semi-autonome** considérée comme un sous-système
- Offre un ensemble de fonctionnalités cohérentes entre elles
- Ces fonctionnalités sont **implémentées** par les classes du composant et exposées sous forme d'**interfaces requises** ou **offertes**



# Composant



(2/3)

- Deux origines:
  - Des composants métiers propres à une entreprise
  - Des composants disponibles sur le marché
    - Exemples : APIs, bibliothèques et packages prédéfinis

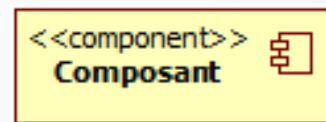
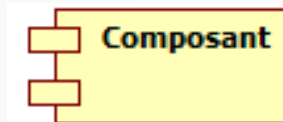
# ► Composant

(3/3)

- Caractérisé par :
  - un nom
  - une spécification externe sous forme
    - 1..\* interfaces *fournies* ou *offertes*
      - interfaces proposées par le composant aux autres composants
    - 0..\* interfaces *requises*
      - interfaces nécessaires au bon fonctionnement du composant

- Formalisme :

UML 1



UML 2



# Représentation

- Deux types de représentation :
  - « *Boîte blanche* »
    - Vue interne du composant qui décrit son implémentation à l'aide de classificateurs (classes, autres composants) qui le composent
  - « *Boîte noire* »
    - Vue externe du composant qui présente ses interfaces fournies et requises sans entrer dans le détail de l'implémentation du composant

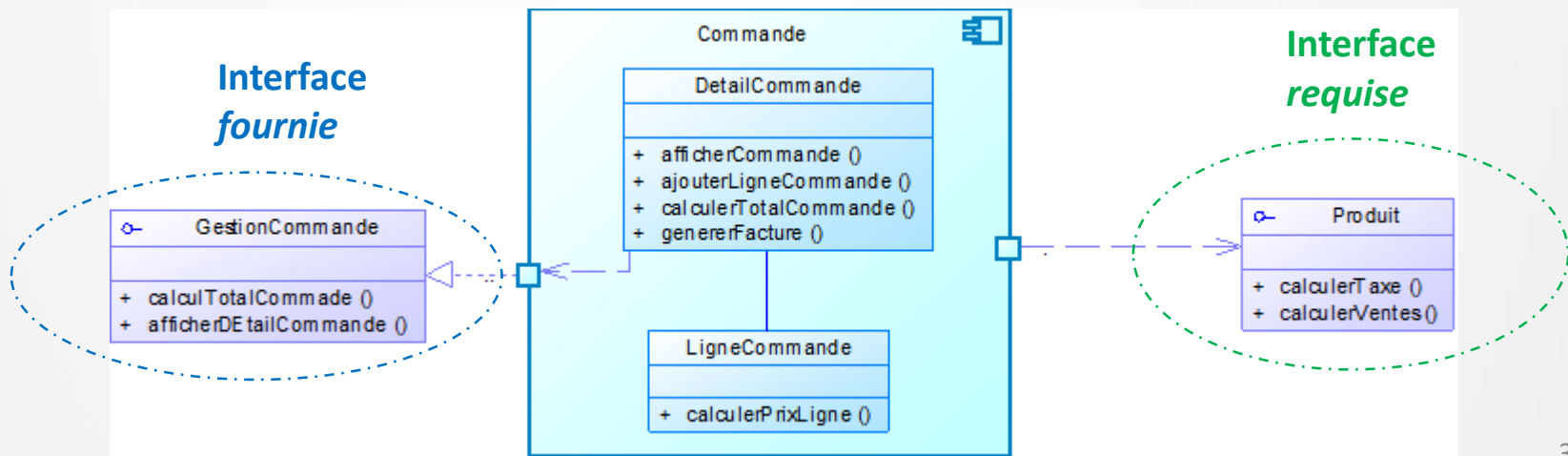


# Représentation en boîte blanche

# ► Composant : Boîte blanche

(1/7)

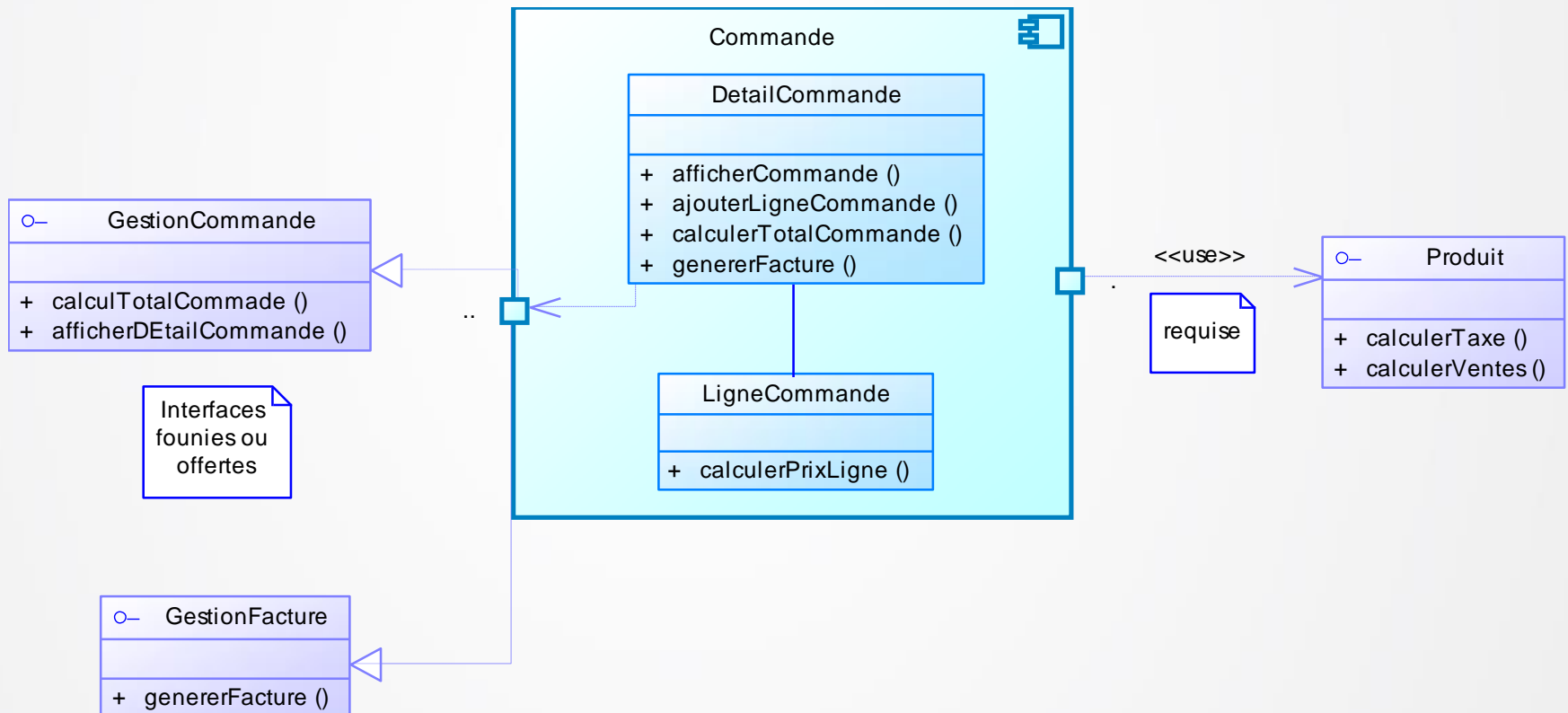
- Représentation 1 :
  - Les classes des composants sont visibles
  - Les interfaces fournies et/ou requises sont visibles
  - Le port relie le composant et les interfaces et est représenté par un petit carré sur les composants
- Exemple 1 :



# Composant : Boîte blanche

(2/7)

- Exemple 2 avec deux interfaces fournies :

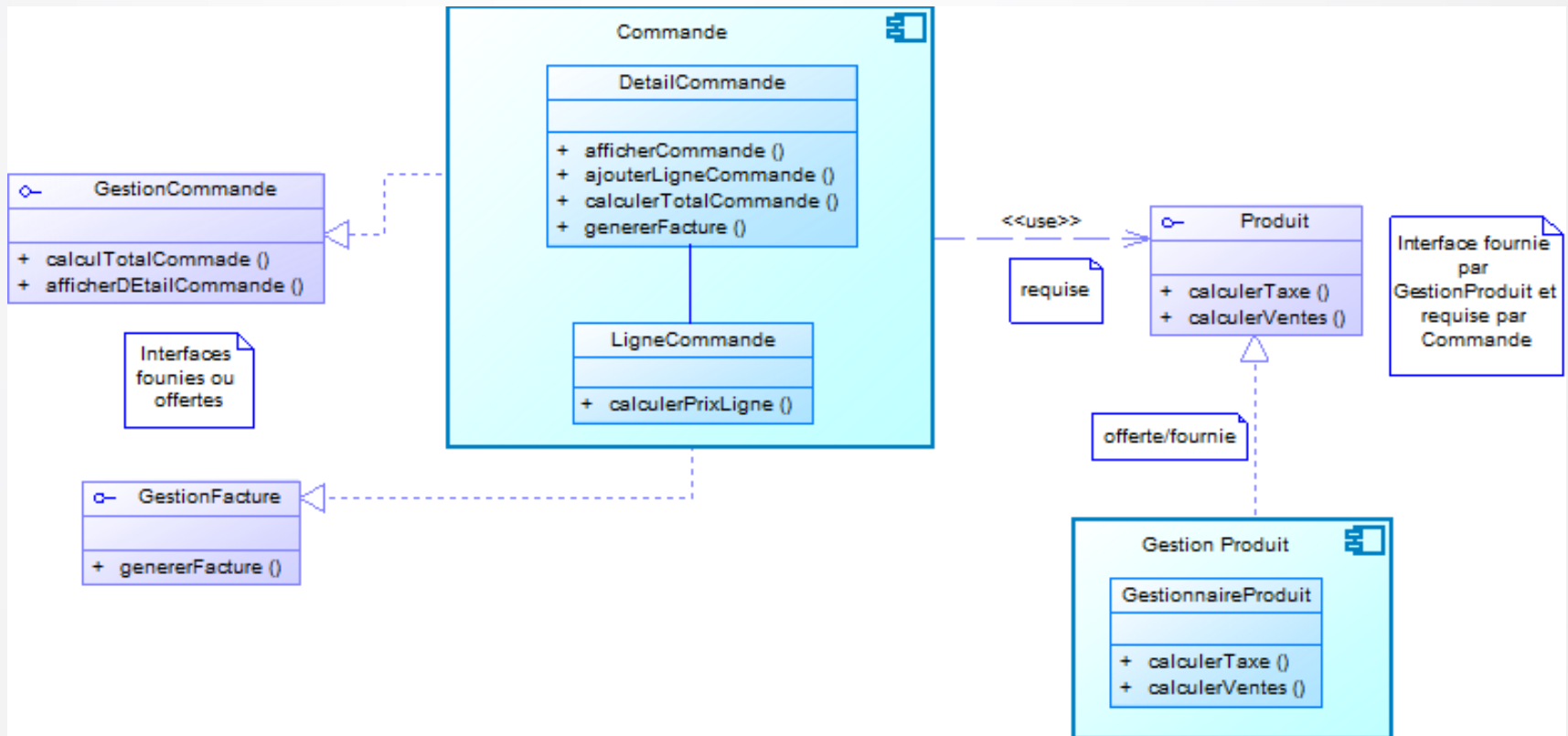




# ► Composant : Boîte blanche

(3/7)

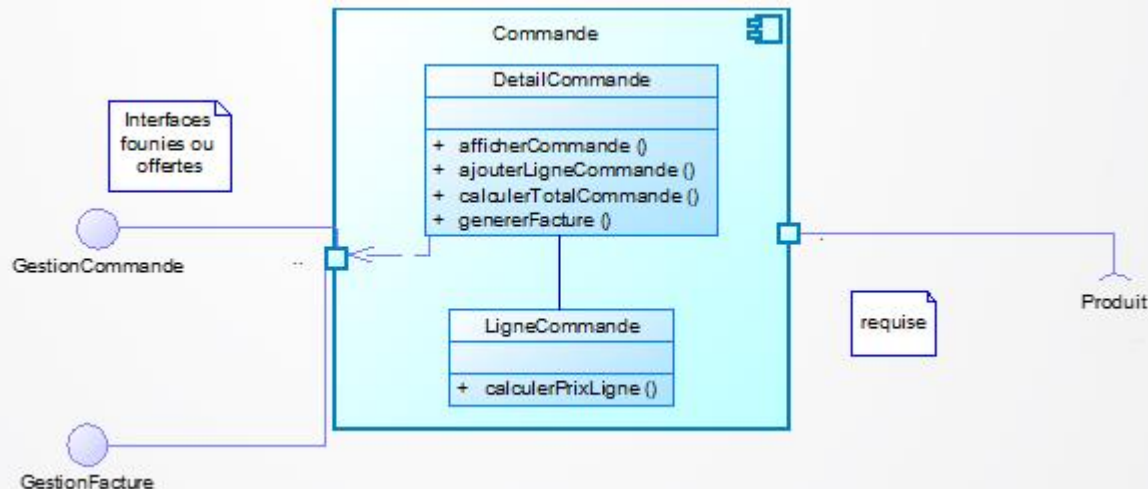
- Exemple 3 avec deux composants :



# ► Composant : Boîte blanche

(4/7)

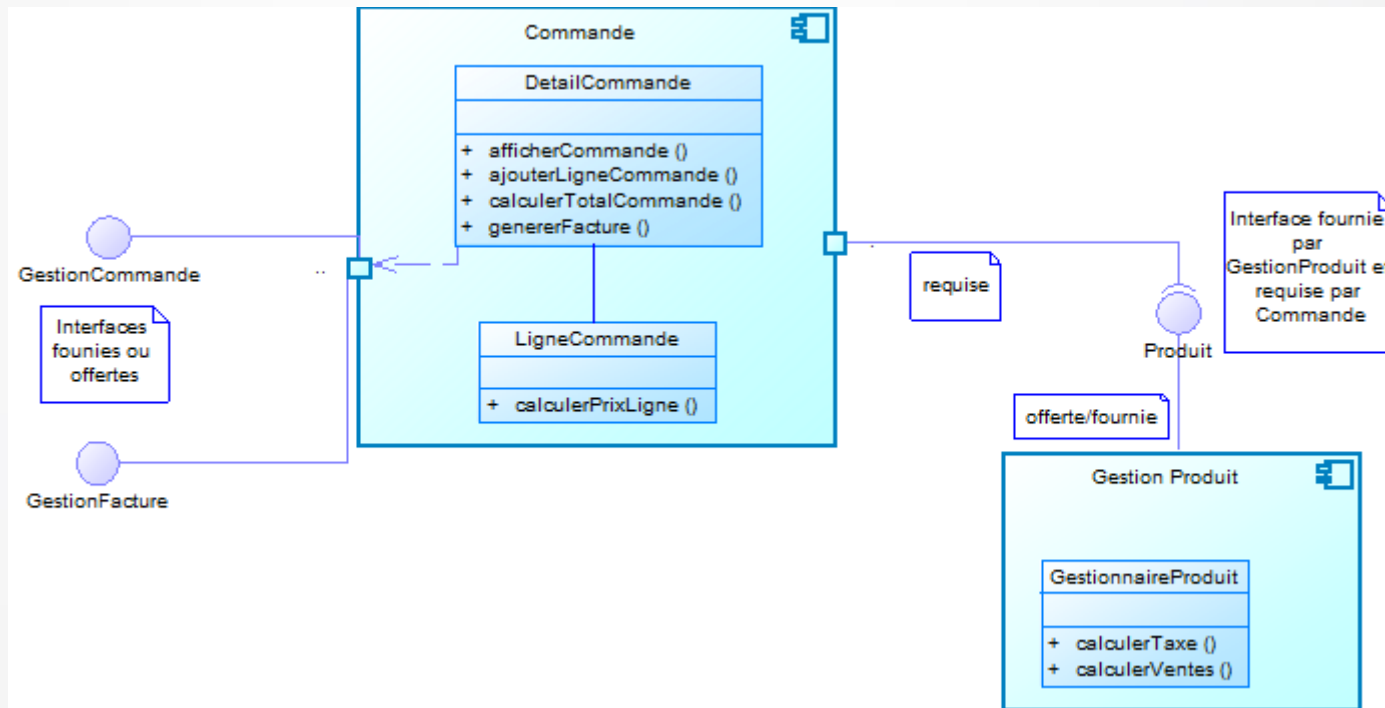
- Représentation 2 :
  - Les classes des composants sont visibles
  - Les interfaces requises ou fournies sont sous forme de *lollipops*
- Exemple 1 :



# ► Composant : Boîte blanche

(5/7)

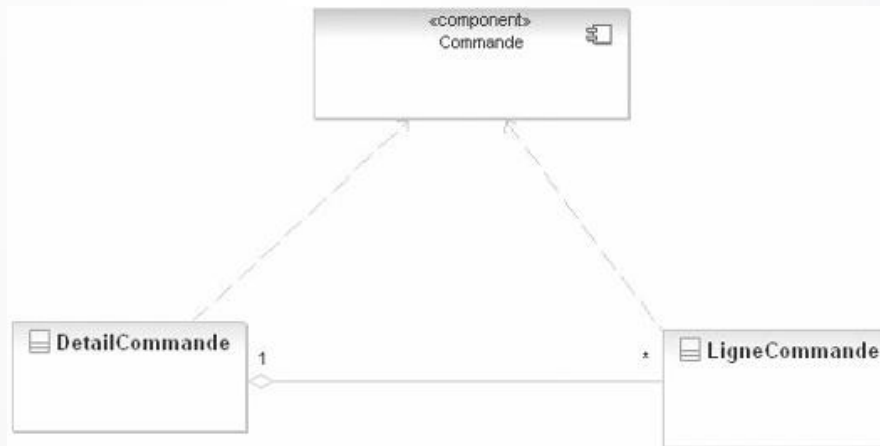
- Exemple 2 avec deux composants :



# ► Composant : Boîte blanche

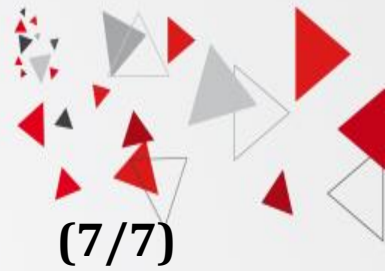
(6/7)

- Représentation 3 : Via des dépendances
  - Liens de dépendance entre le composant et les classificateurs qui le composent
  - Relations présentes entre les classificateurs (association, composition, agrégation)
    - Exemple :



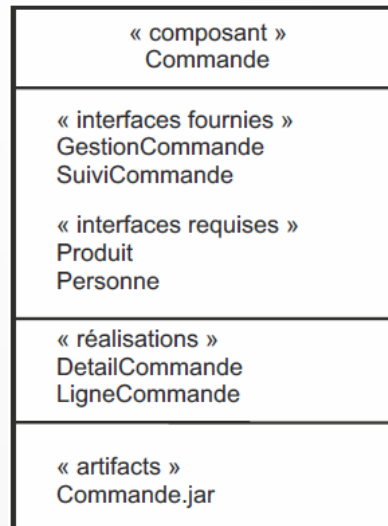
Source : UML2 analyse et conception

# ► Composant : Boîte blanche



(7/7)

- Représentation 4 : En compartiments
  - Trois compartiments:
    - Pour les interfaces fournies et requises
    - Pour les classificateurs (classes, autres composants)
    - Pour les artefacts (élément logiciel : jar, war, ear, dll) qui représentent physiquement le composant
      - Exemple :



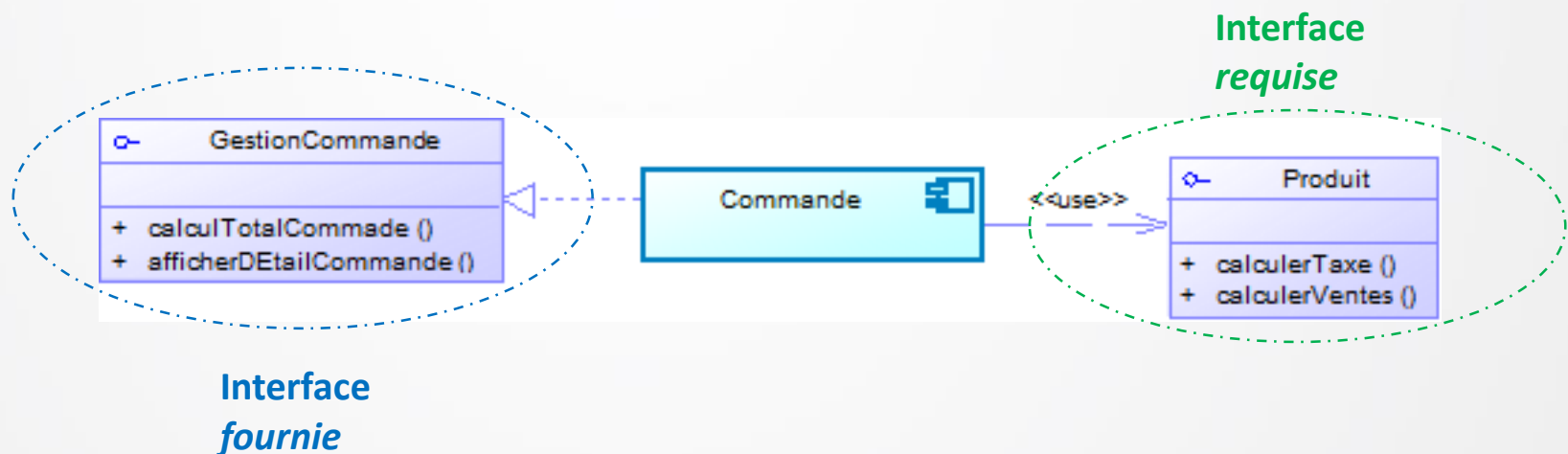


# Représentation en boîte noire

# ► Composant : Boîte noire

(1/5)

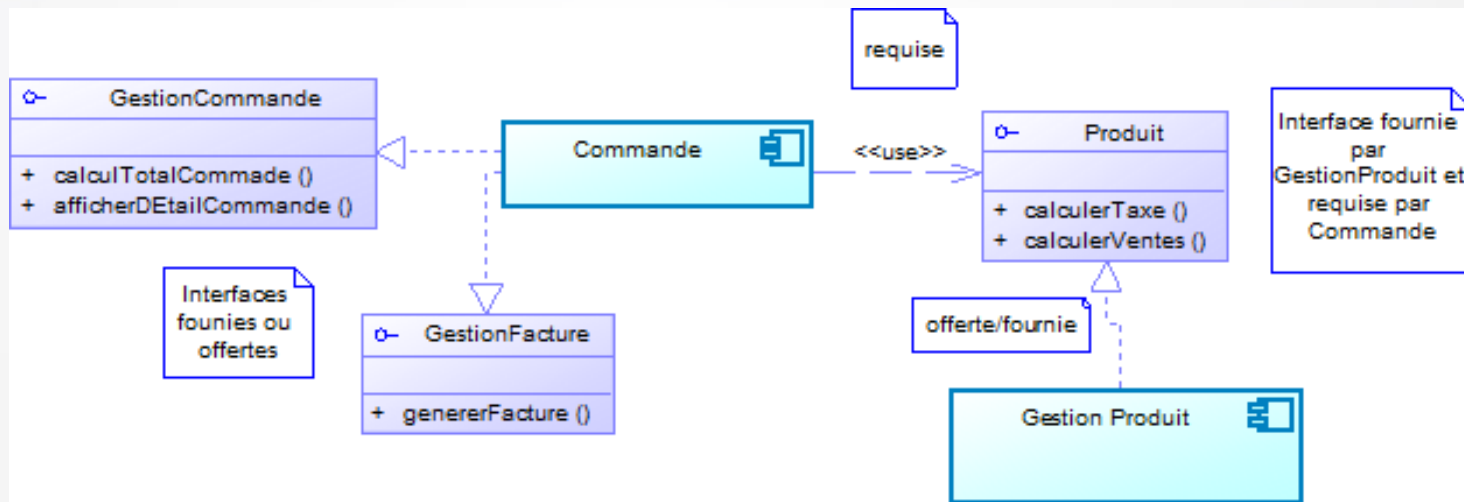
- Représentation 1 :
  - Utilisation des dépendances d'interfaces *use* et *realize* :
    - « *realize* » pour une interface *fournie* ou *offerte*
    - « *use* » pour une interface *requis*
- Exemple 1 :



# ► Composant : Boîte noire

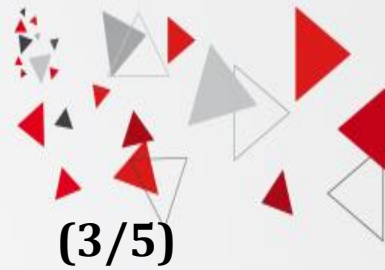
(2/5)

- Exemple 2 avec deux composants :





# ► Composant : Boîte noire



(3/5)

- Représentation 2:
  - Connecteurs d'assemblage ou *lollipops* :
    - un trait et un cercle pour une interface *fournie*
    - un trait et un demi-cercle pour une interface *requise*
- Exemple 1 :

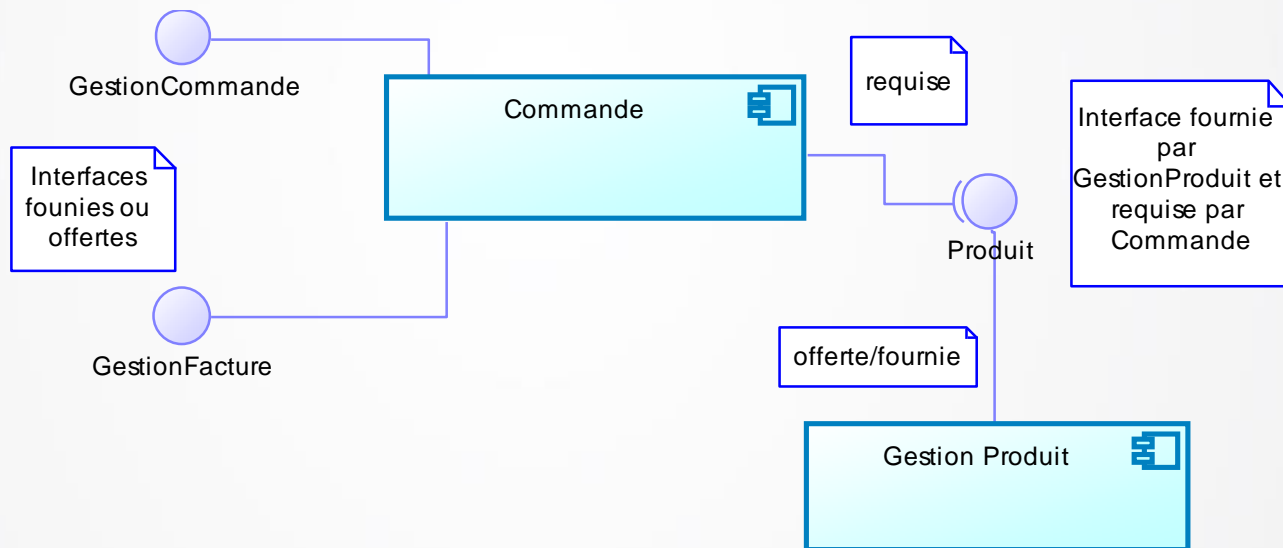


Source : UML2 analyse et conception

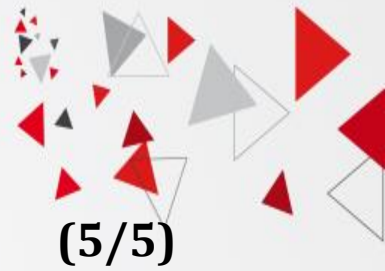
# ► Composant : Boîte noire

(4/5)

- Exemple 2 avec deux composants :

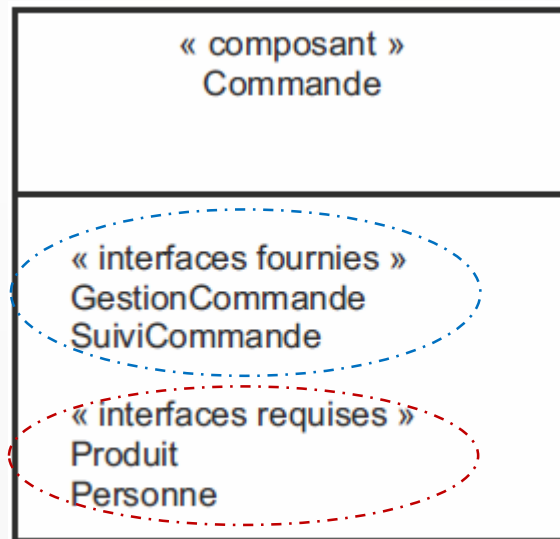


# ► Composant : Boîte noire



(5/5)

- Représentation 3 : En compartiments
  - Décrire sous forme textuelle les interfaces fournies et requises à l'intérieur d'un second compartiment
- Exemple :

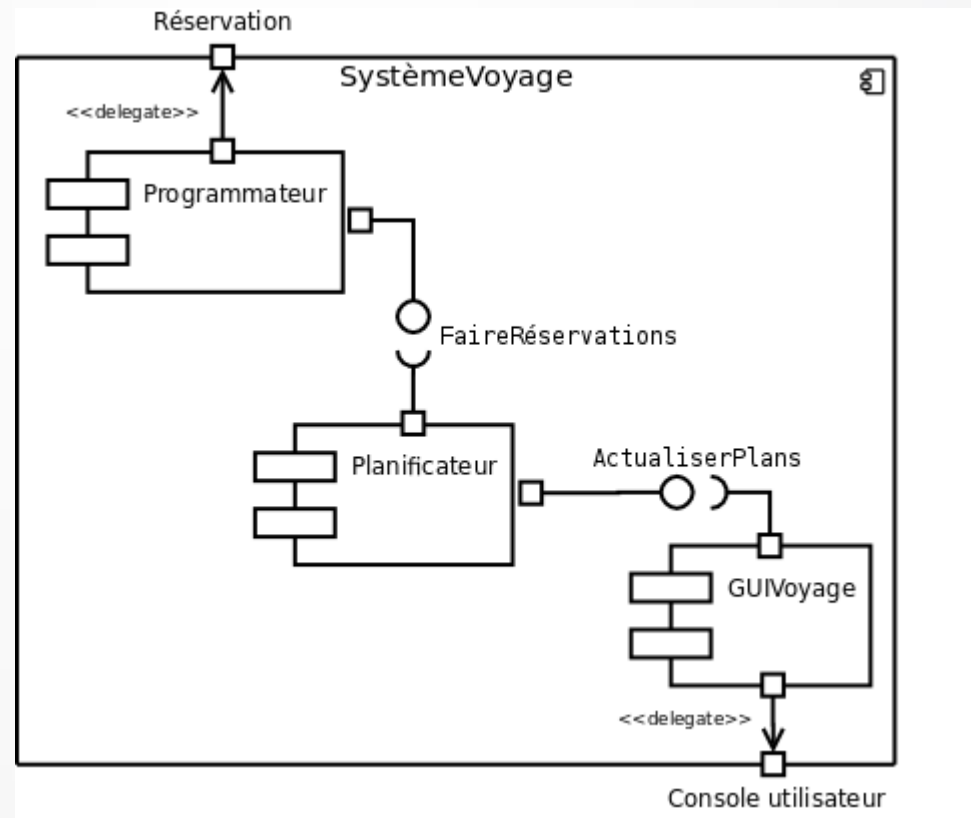




Remarque : Composant complexe

# ► Composant complexe

- Un composant peut être constitué de composants
  - Exemple:





# Diagramme de déploiement



# Présentation

- Représente l'architecture physique du système :
  - Ensemble de nœuds
    - Correspondent aux supports physiques (serveurs, routeurs...)
  - Connexions entre les nœuds
  - Répartition des artefacts sur chaque nœud



Nœud

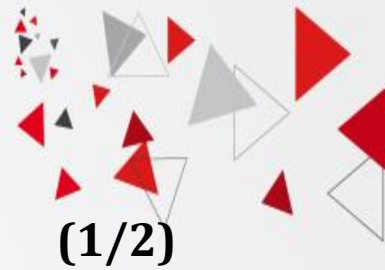




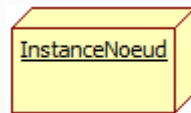
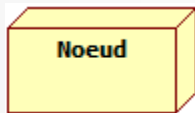
# Nœud

- Élément principal du diagramme
- 2 types :
  - Device : physique
  - Environnement d'exécution ou Execution environment : logique

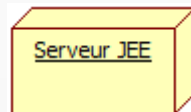
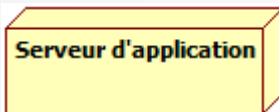
# ► Nœud « device »



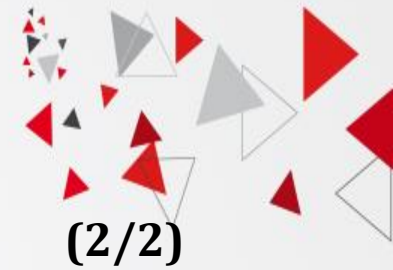
- Ressource matérielle de traitement : « **Device** »
  - Exemples : PC, smartphone, tablette, Serveur
- Peut posséder des attributs
  - Exemples : CPU, RAM,...
- Les artefacts sont mis ou **déployés** sur les nœuds
- Formalisme :
  - Nœud et instance d'un nœud



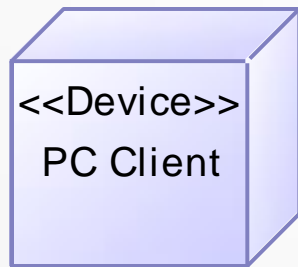
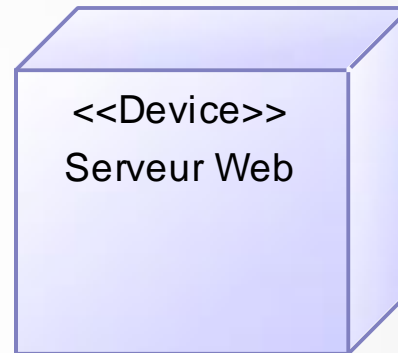
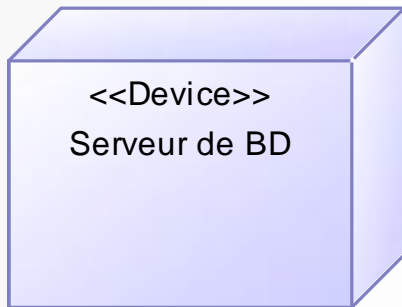
- Exemple 1 :



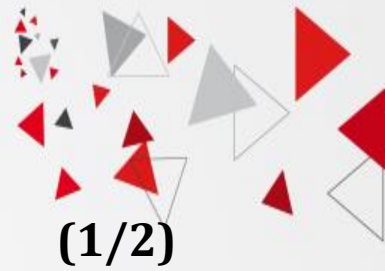
# ► Nœud « device »



- Exemples :



# ► Environnement d'exécution

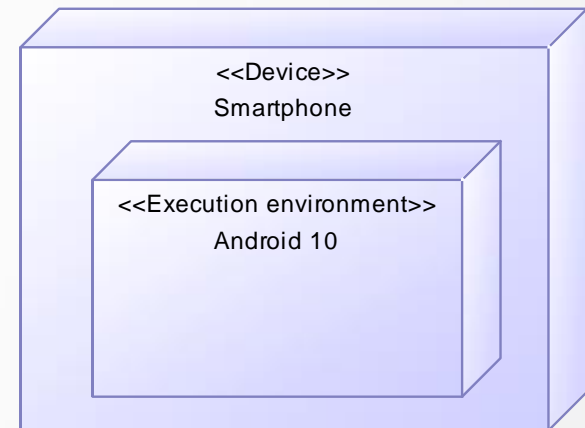
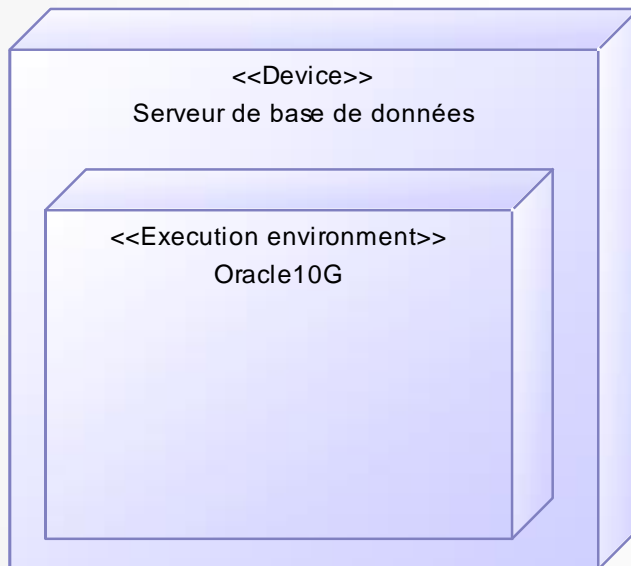


- Nœud qui représente un serveur **logiciel**
  - Serveur web
    - Exemples : Apache, IIS
  - Conteneur web
    - Exemples : Tomcat, Wamp, EasyPhp
  - Serveur de bases de données
    - Exemples : Oracle10g, MySQL
- Est représenté par un nœud « **Execution environment** »
- Doit être toujours mis sur nœud **physique**

# ► Environnement d'exécution

(2/2)

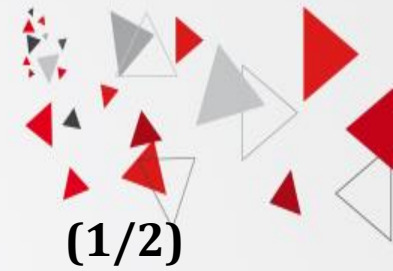
- Exemples :



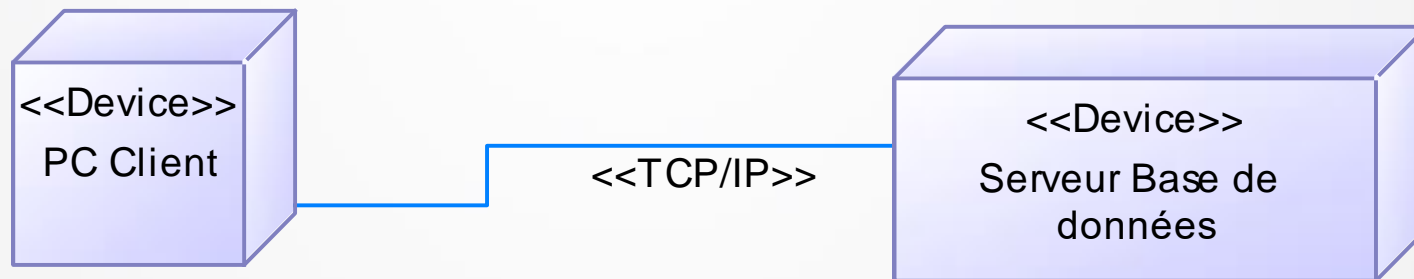


# Connexions

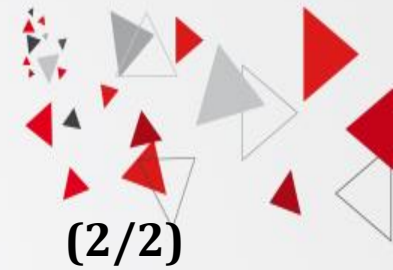
# ► Connexions entre nœuds



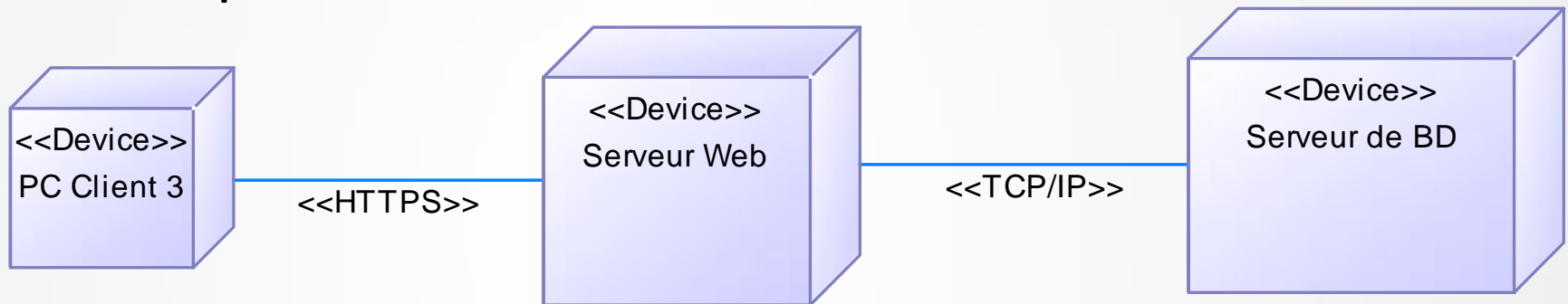
- Permettent l'échange d'informations entre nœuds
- Sont caractérisés par des protocoles de communication
  - Exemples : TCP/IP, Ethernet, HTTP, HTTPS
- Exemple 1 :



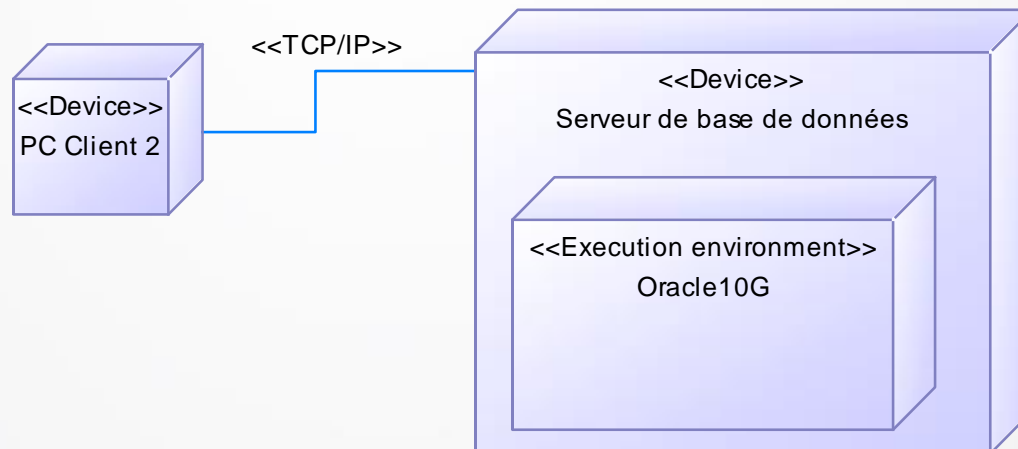
# ► Connexions entre nœuds



- Exemple 2 :



- Exemple 3 :





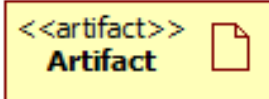


# Artefact


# ► Artefact



- Représente un élément **physique** et **concret** utilisé ou produit
- Est **déployé** sur un nœud
- Exemples :
  - \*.exe, \*.dll, \*.jar, \*.war

- Formalisme : 

- Peut être relié à d'autres artefacts par des liens de dépendance

- Exemple : 

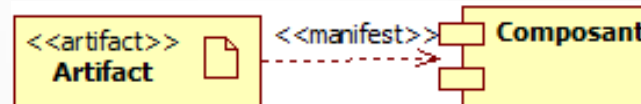


# « *Manifest* »

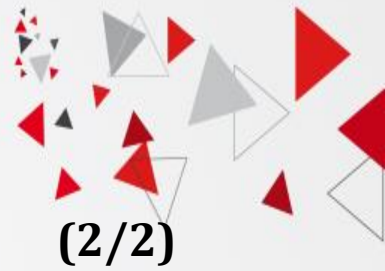
# ► Artefact et composant

(1/2)

- Un artefact peut représenter (implémenter) des composants ou d'autres classificateurs
- Un composant peut être manifesté par plusieurs artefacts déployés dans des nœuds différents
- Formalisme :



# ► Artefact et composant



(2/2)

Composant	Artifact
Abstrait	Concret (api, web service...)
Non exploitable (non utilisable)	Exploitable
Est manifesté par un ou plusieurs artifacts	Est déployé sur un nœud
Peut être réutilisé au niveau conceptuel	Peut être réutilisé au niveau de l'implémentation



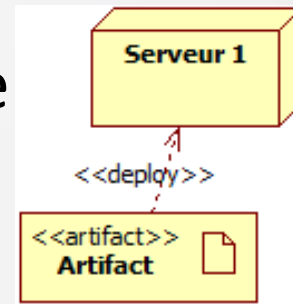
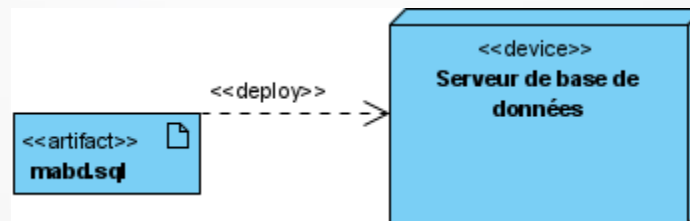
« *Deploy* »

# ► Nœud et artefact : déploiement

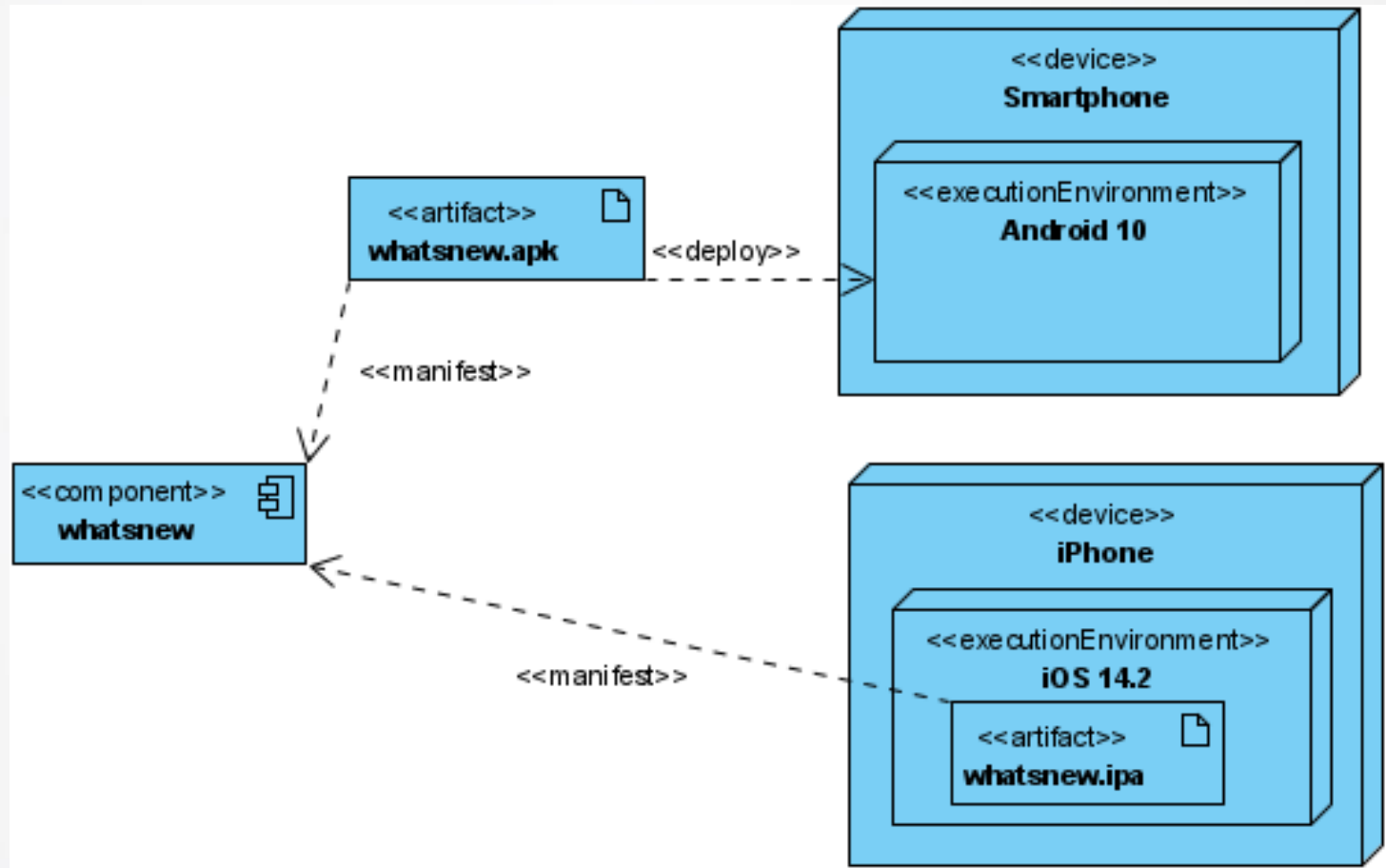
- Deux représentations :
  - Représentation inclusive
    - Exemple :



- Représentation avec un lien de dépendance typé «deploy»
  - Exemple :

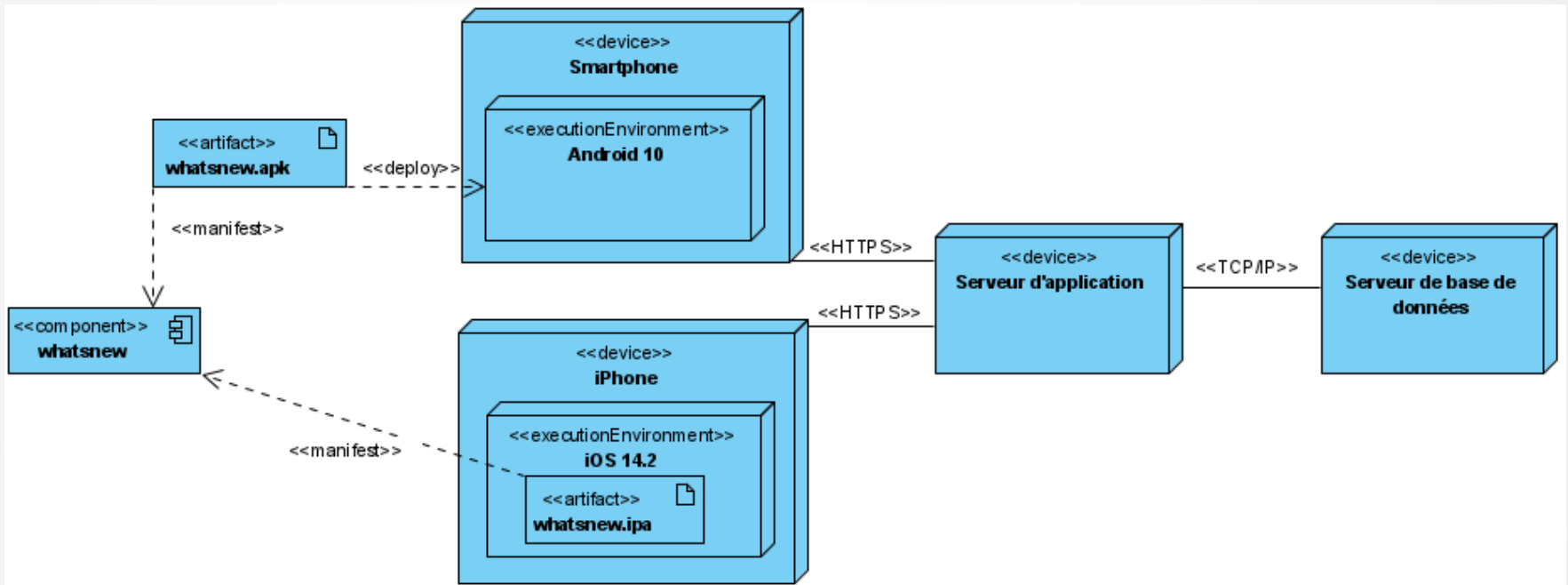


# ▶ Example 1

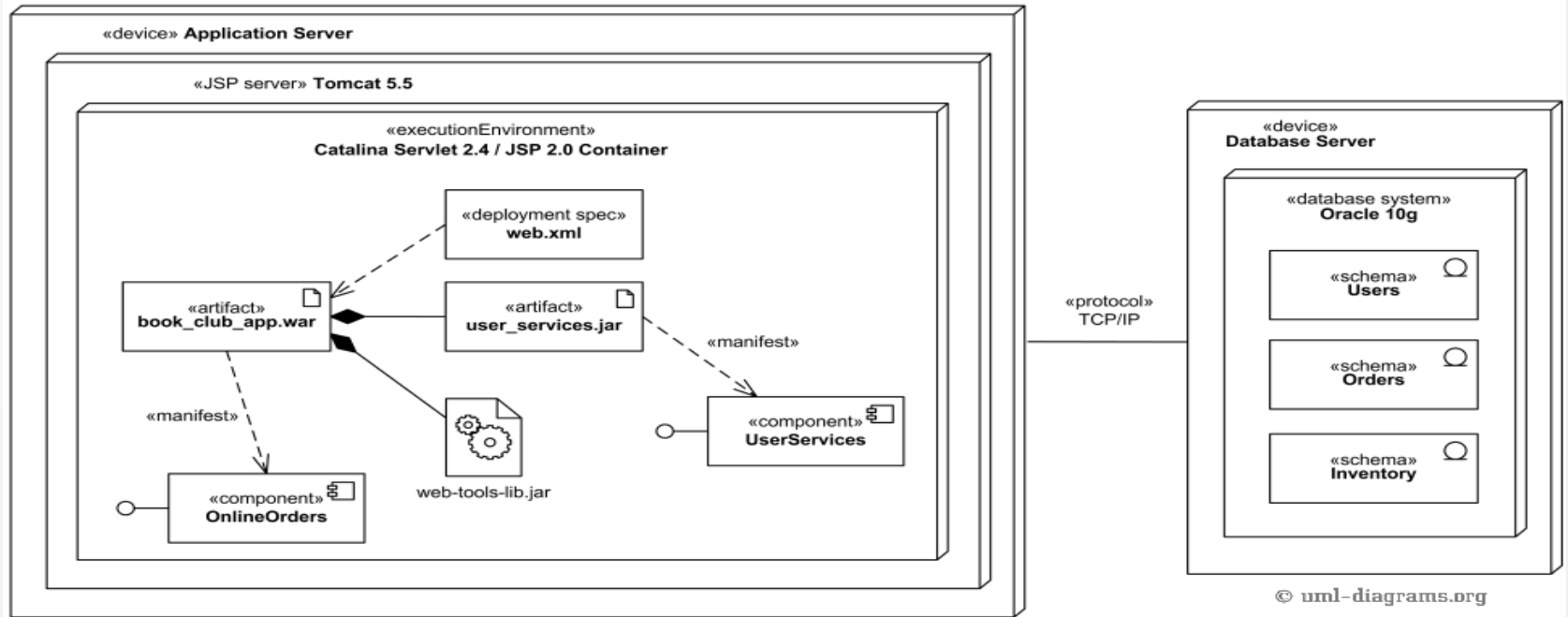




# ▶ Exemple 2



# ▶ Example 3



# ► Des questions?

