R5.A.08 :
Qualité de développement

Chouki Tibermacine
Chouki.Tibermacine@univ-ubs.fr

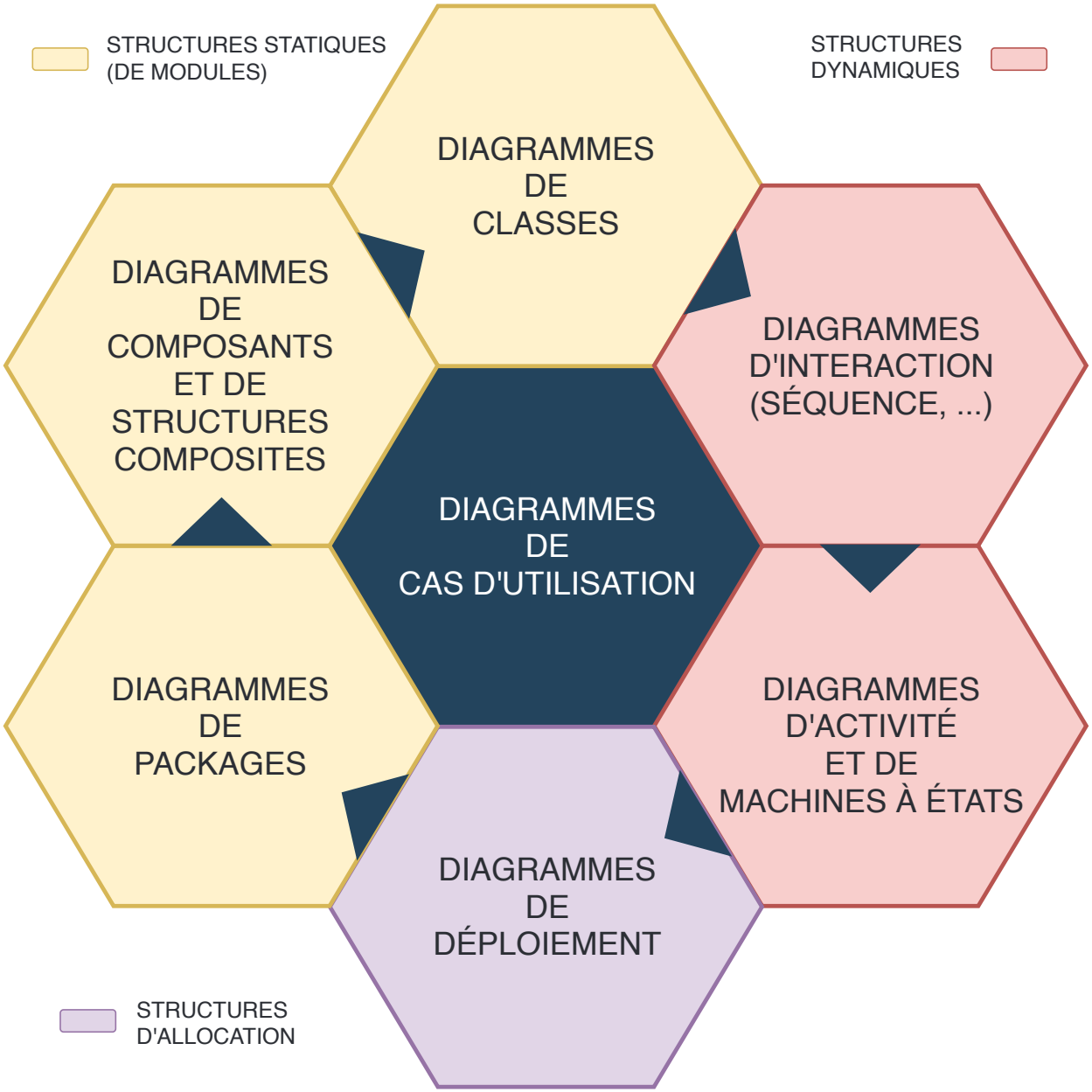
Plan prévisionnel de la ressource

1. Intro aux architectures logicielles
2. Documentation d'architectures en UML
3. Styles et patrons d'architectures
4. Architectures à microservices
5. Design & Implem de frameworks
6. Patrons de conception : Façade, Bridge, MVC et MVVM
7. Patrons de conception : Builder, Proxy et Visitor
8. Autres patrons

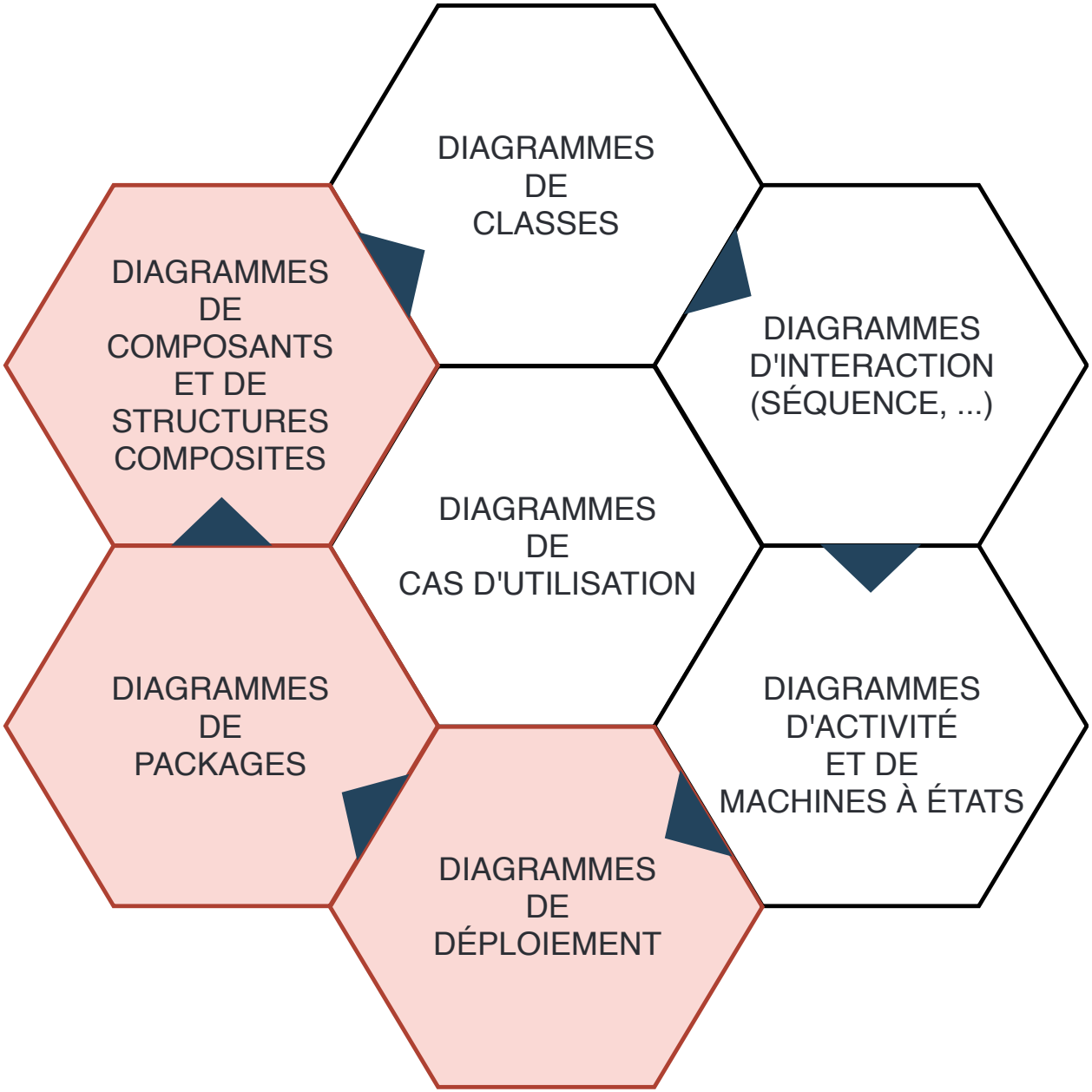
Plan prévisionnel de la ressource

1. Intro aux architectures logicielles
2. Documentation d'architectures en UML
3. Styles et patrons d'architectures
4. Architectures à microservices
5. Design & Implem de frameworks
6. Patrons de conception : Façade, Bridge, MVC et MVVM
7. Patrons de conception : Builder, Proxy et Visitor
8. Autres patrons

Différentes structures possibles



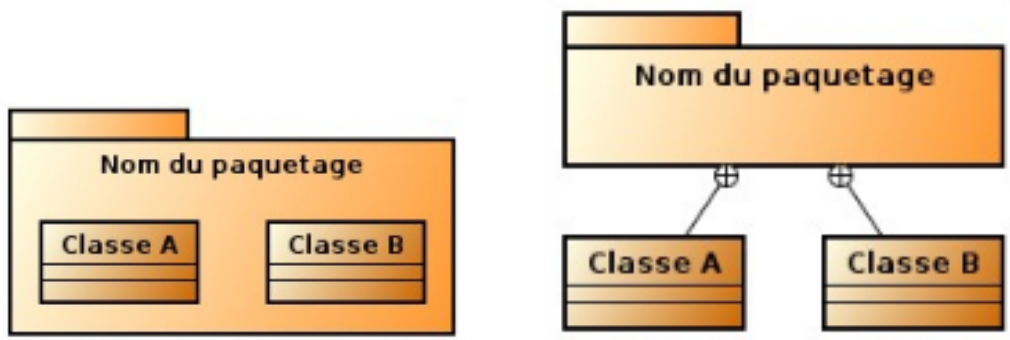
Ce que nous allons voir



Diagrammes de packages

Diagrammes de packages (ou de paquetages)

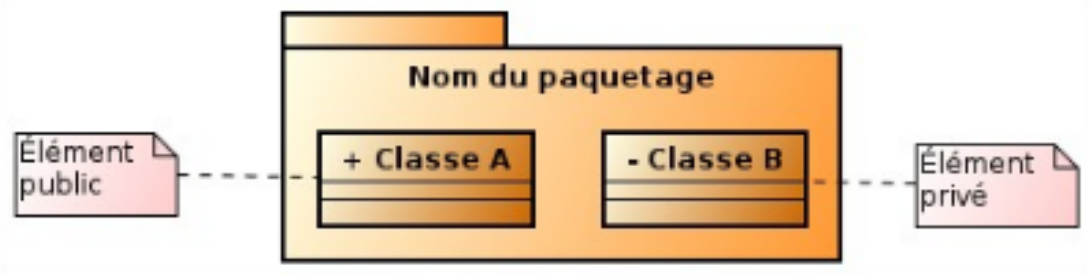
- Permettent de modéliser l'organisation des packages et de leurs constituants (classes, activités, cas d'utilisation, ...)
- ça modélise des espaces de noms (éléments sémantiquement liés, avec des noms uniques au sein de l'espace de noms)
- C'est représenté par un dossier, avec ses éléments à l'intérieur ou à l'extérieur [1]



1. Figures issues de : http://remy-manu.no-ip.biz/UML/Cours/coursUML10.pdf

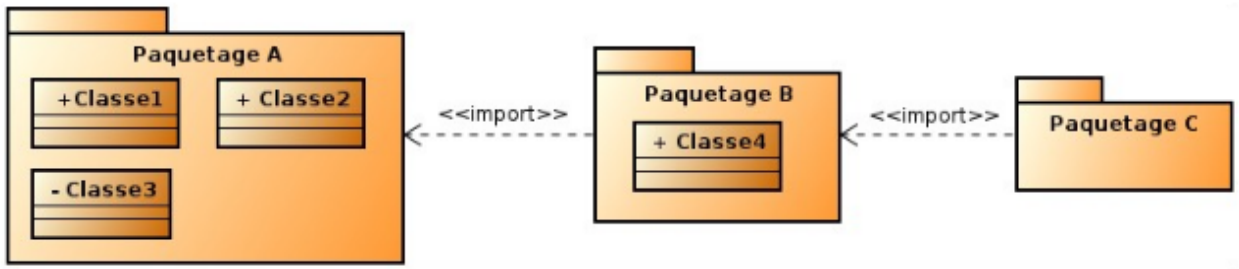
Visibilité dans les diagrammes de packages

- Chaque élément dans un package est soit privé ou public
- Élément public : accessible aux éléments dans les autres packages
- Élément privé : n'est accessible qu'aux éléments dans le même package



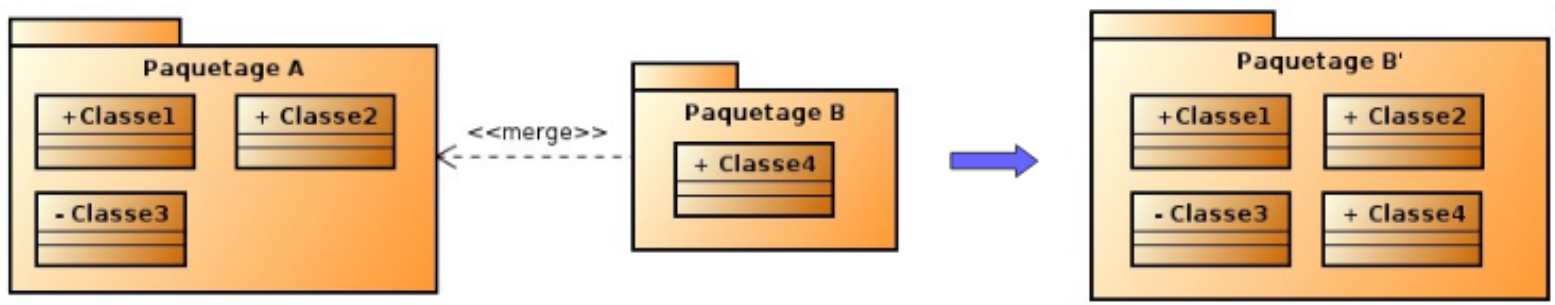
Dépendances entre packages

- Deux sortes de dépendances entre packages : *import* et *merge*
- Dépendances de type “*import*” :
  - Importer d’un package tous les éléments publics
  - Rendre publics tous les éléments publics importés pour les autres packages (export implicite)



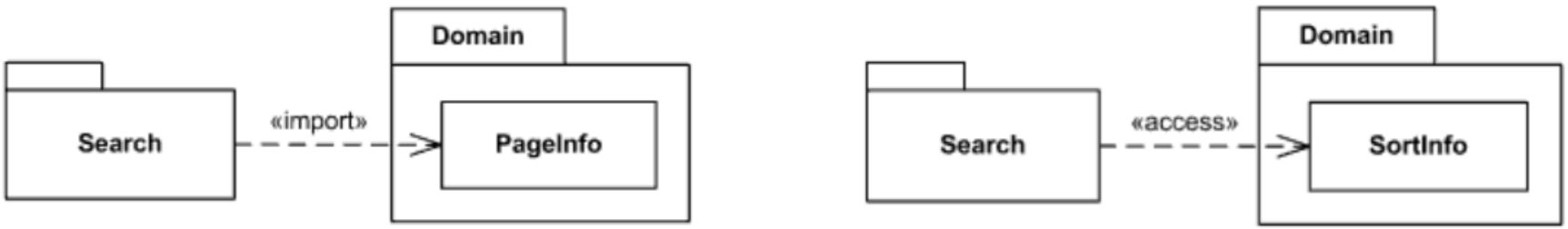
Dépendances entre packages -suite-

- Dépendances de type “*merge*” :
  - Correspond à la fusion d’un package avec un autre (fusion dans un seul sens : la source intègre les éléments de la cible)

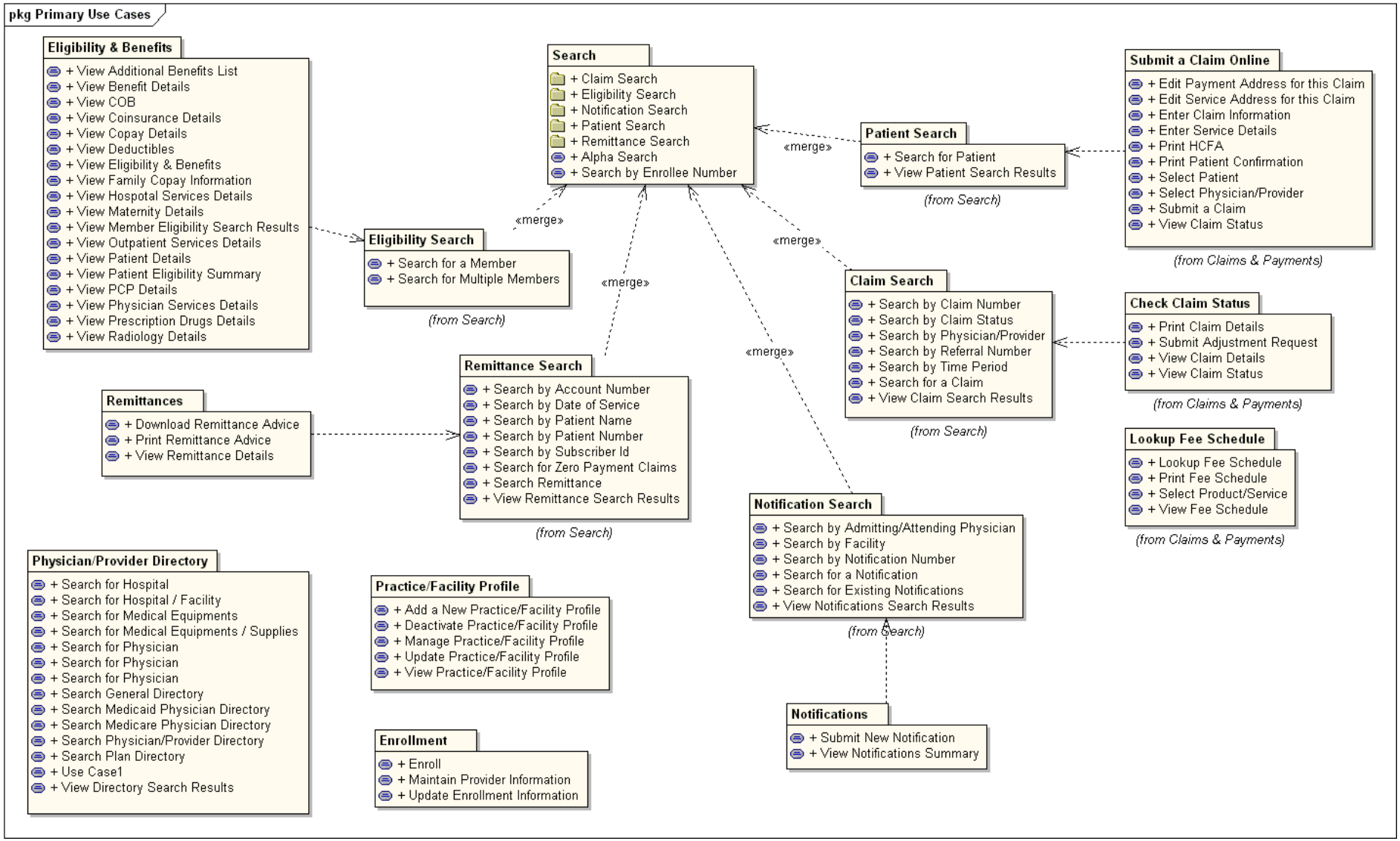


Import public ou privé d'un élément dans un package

- Import **public** d'élément dans un package : l'élément importé sera ajouté à l'espace de noms et rendu visible à l'extérieur
- Import **privé** (`access`) d'élément : l'élément est ajouté mais pas exposé



Exemple de diagramme de packages (avec des UC)



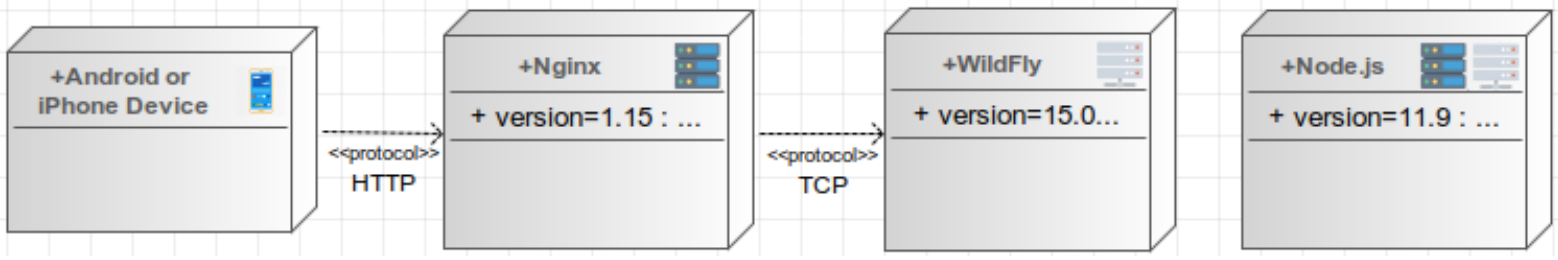
Quand utiliser les diagrammes de packages ?

- Grouper les use cases pour modéliser les fonctionnalités d'un système de grande taille
- Modéliser les éléments majeurs (sous-systèmes, couches, ...) d'un système de grande taille et leurs dépendances
- Les diagrammes de packages représentent un groupement statique d'éléments (ce n'est pas lié à l'exécution)

Diagrammes de déploiement

Diagrammes de déploiement : les noeuds

- Modéliser l'architecture à l'exécution d'un système
- Elle inclut :
  1. la configuration des éléments d'infrastructure (**nodes**)
  2. et l'affectation des éléments logiciels concrets (**artifacts**) à ceux-ci
- Deux sortes de nodes : des **devices** (matériel) ou des **execution environments** (infra logicielle)
- Les nodes sont connectés par des voies de communication (**communication path**), des associations UML

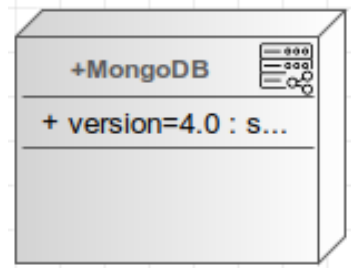


Diagrammes de déploiement : les artefacts

- Les artifacts sont une **manifestation** (concretisation physique) d'éléments logiciels (ex : composants)
- Exemple : une archive JAR

Diagrammes UML et stéréotypes

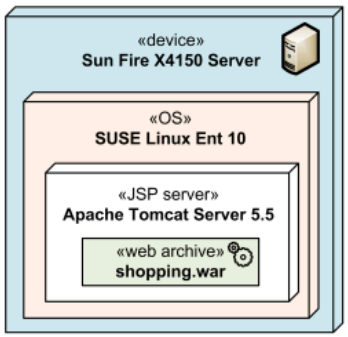
- Un stéréotype est un moyen de donner plus de sémantique à un élément de modélisation (noeud, artefact, lien, ... ou tout autre élément UML)
- Indiquer qu'un noeud est un *device* ou un serveur de base de données



- Dire qu'un artefact est un *module EJB*, un programme Node.js ou un fichier de configuration de déploiement (.yml par ex)
- Un stéréotype peut avoir une apparence graphique (icône, comme ci-dessus) ou une annotation textuelle (Ex : « protocol »)
- Stéréotypes faciles à manipuler sur les IDE

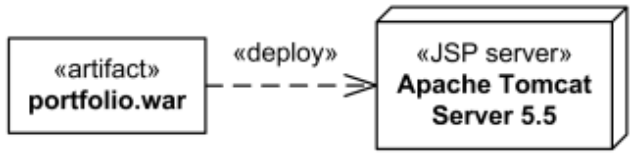
Noeuds hiérarchiques

- Définis par des associations de type composition ou bien en utilisant des structures internes (composites) d'un noeud
- Les "environnements d'exécution" (un OS, une VM, un conteneur, ...) sont des noeuds qui font partie de la structure interne d'un noeud représentant le matériel (un "device")
- On modélise souvent le fait que les artefacts sont déployés dans des noeuds de type "environnement d'exécution"
- Dans UML, pas de stéréotypes standards, mais il existe des stéréotypes non-normatifs : «application server», «client workstation», «mobile device», «embedded device», ...

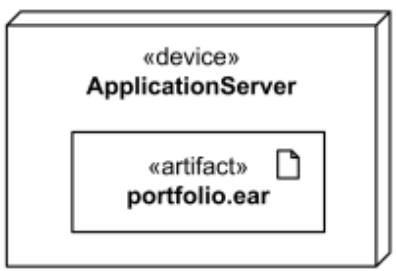


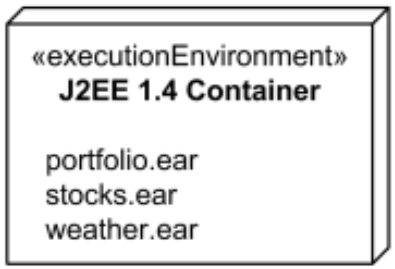
Déploiement d'artefacts

- Le déploiement est une dépendance UML stéréotypée « deploy »

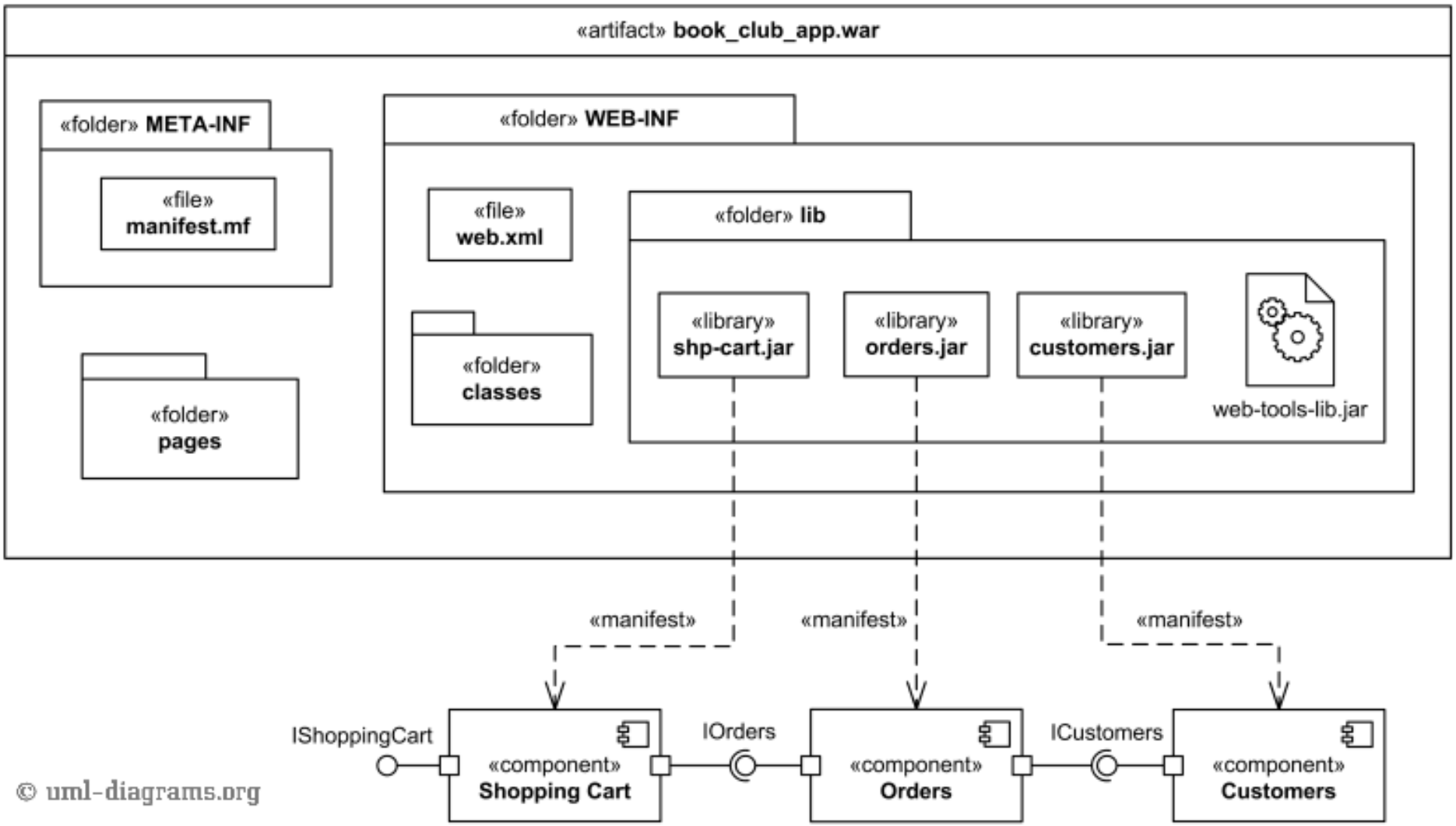


- ça peut être défini aussi dans la structure interne ou comme attribut

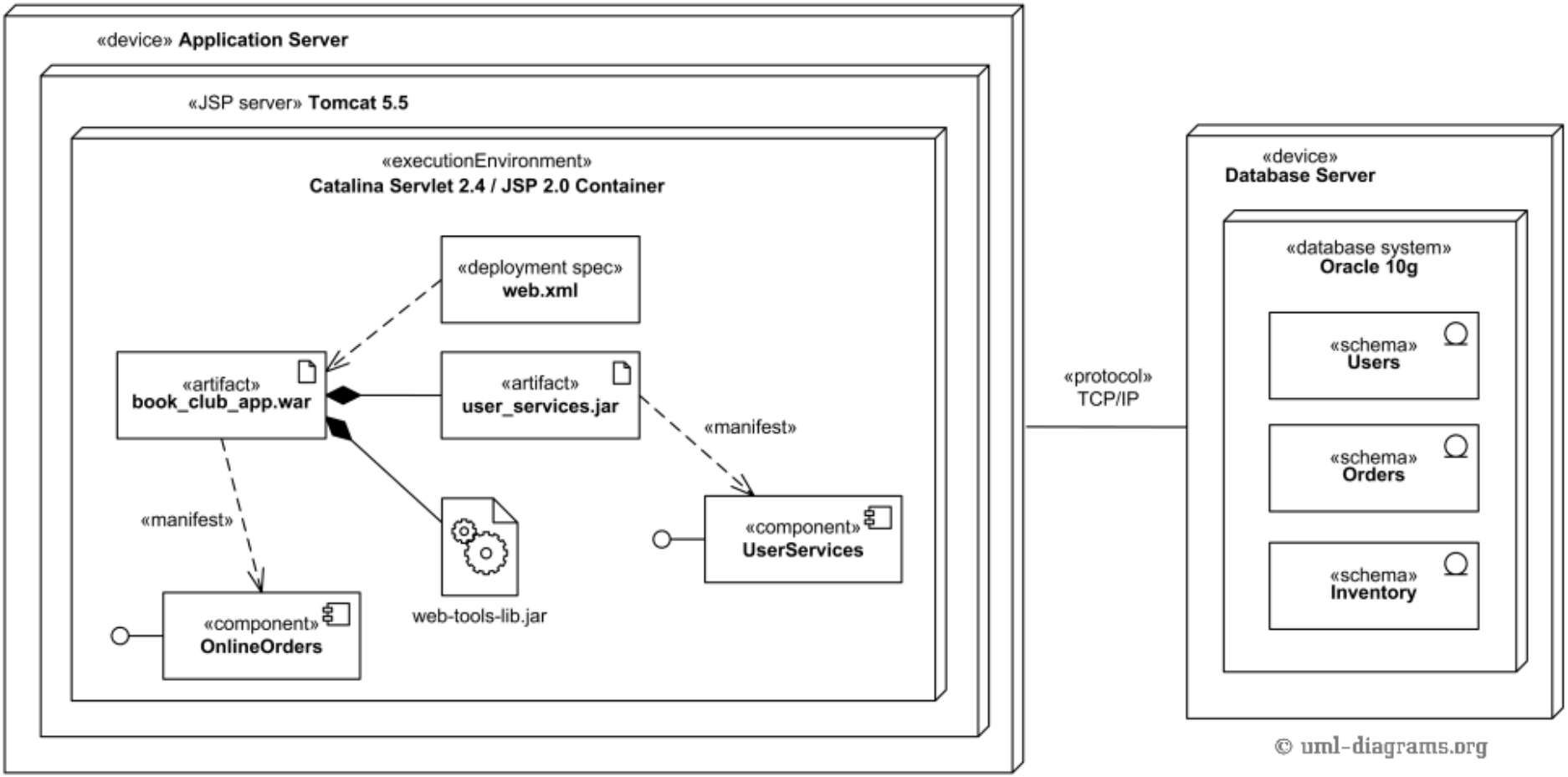




Exemple de diag. de déploiement (manifestation)



Exemple de diag. de déploiement

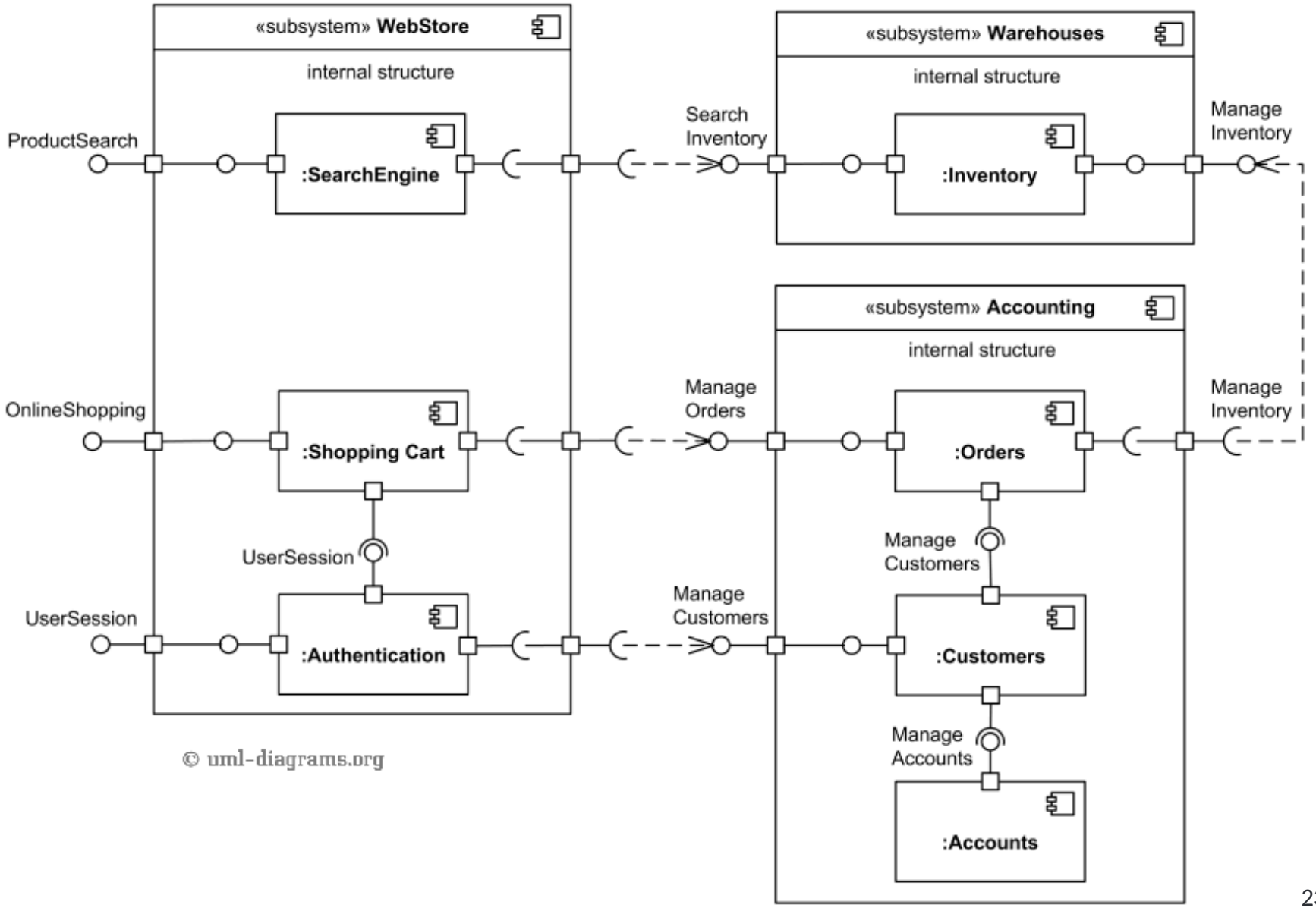


Diagrammes de composants

Diagrammes de composants

- Modélisent la structure d'un logiciel en termes de sous-systèmes, la décomposition de chacun et leurs dépendances
- Notions de **composants**, d'**interfaces requises** et **fournies**, de **ports** et de **connecteurs**
- Un composant est vu ici comme une entité modulaire et réutilisable dans le système, qui explicite ses dépendances
- Un composant est vu aussi comme une entité ayant une structure interne explicite, ce qui la rend facilement configurable/adaptable

Exemple de diagramme de composants



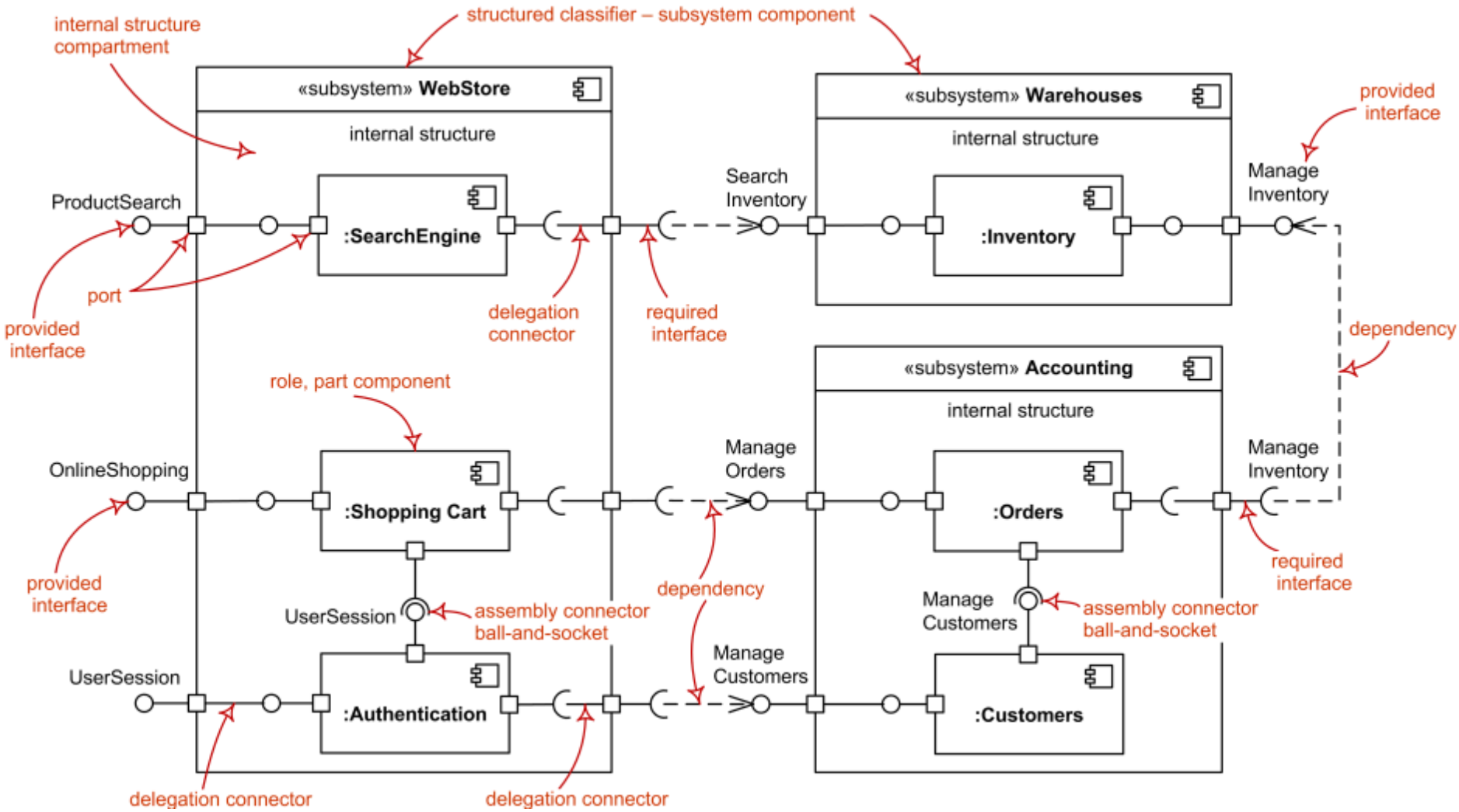
Un composant, comme élément de modélisation

- En UML, c'est une sorte de classe qui représente une partie modulaire d'un système, facilement substituable
- Il expose des interfaces fournies (opérations réalisées par le composant) et des interfaces requises (opérations nécessaires pour son fonctionnement)
- Des fonctionnalités d'un système peuvent être définies en termes d'assemblages d'instances de composants, en reliant leurs interfaces (en utilisant des connecteurs)
- Cet assemblage peut être embarqué dans un composant : ceci va constituer sa structure interne (composite)

Un composant, comme élément de modélisation -suite-

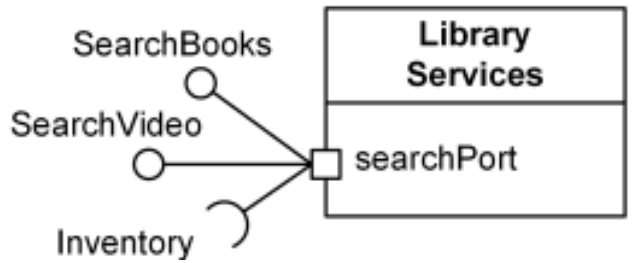
- Un composant = un descripteur (une sorte de classe)
- Ne pas confondre composant (`WebStore` dans l'exemple précédent) et instance de composant (`:ShoppingCart`)
- Les **connecteurs** sont établis entre instances
- Deux types de connecteurs : d'**assemblage** et de **délégation**
- Un **port** : un regroupement d'interfaces (point de communication du composant avec le monde extérieur)
- Un composant se *manifeste* comme un artifact, déployé sur un noeud

Notation graphique

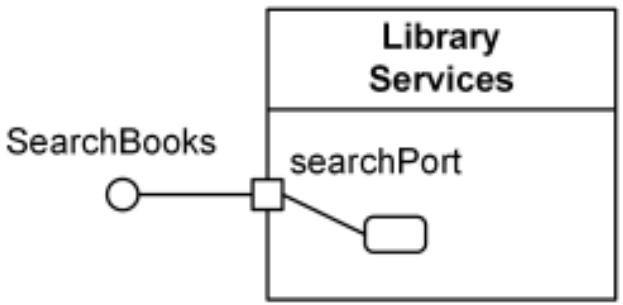


Notion de port

- Un port est le point d'interaction d'un composant avec son environnement ou bien avec ses composants internes (*parts*)
- Notation graphique :

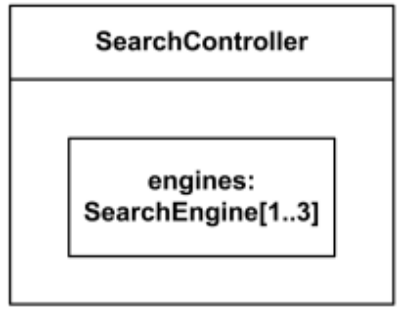


- Un port de comportement : pas de délégation aux composants internes, opérations implémentées directement par le composant (composite), désignées par un état interne

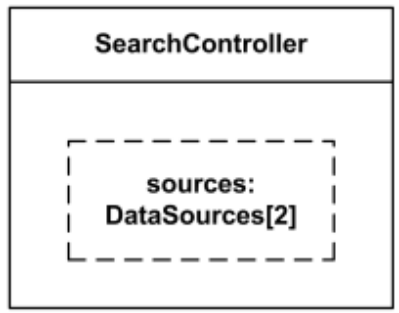


Notion de partie (*part*)

- Une partie (*part* en anglais) désigne une instance interne à un composant
- Notation graphique :

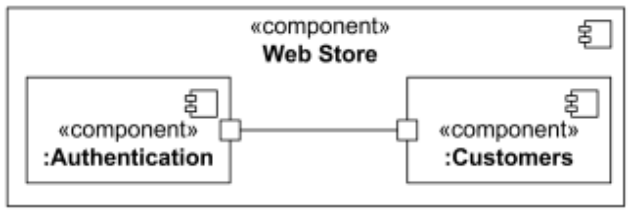


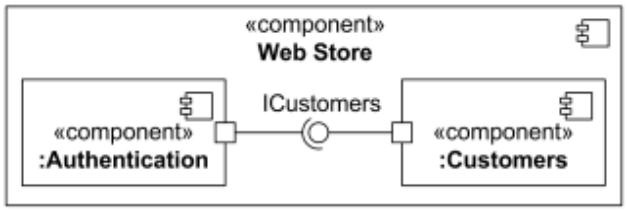
- Une instance peut ne pas appartenir au composant englobant (partagée avec d'autres)



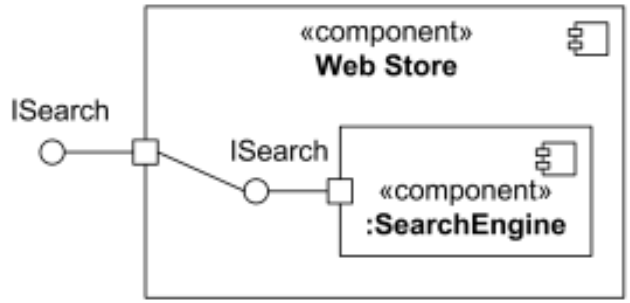
Notion de connecteur

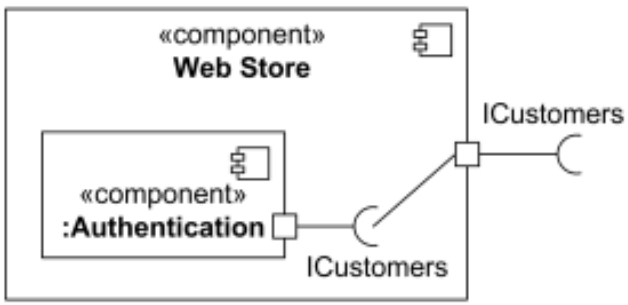
- Un connecteur spécifie un lien qui permet la communication entre des instances de composants
- Connecteur d'assemblage : entre "requis" et "fourni"





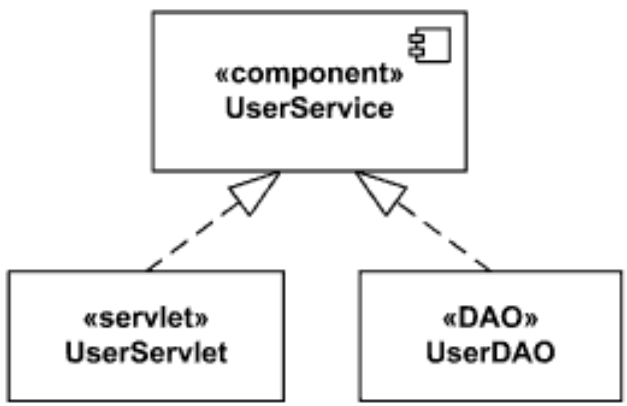
- Connecteur de délégation : fourni-fourni ou requis-requis

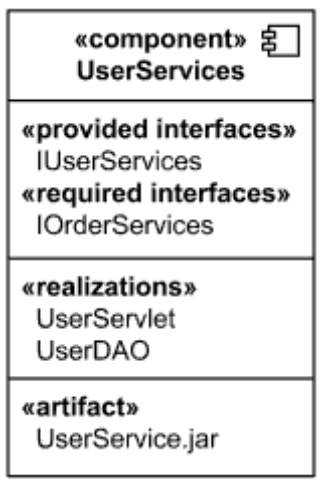




Réalisation de composant

- Un composant peut être réalisé (son comportement est implémenté) par des classes
- Différentes notations possibles :





Diagrammes de structures composites

- Modélisent la structure interne d'un composant