

Applications d'Entreprise avec JEE

Karim Guennoun



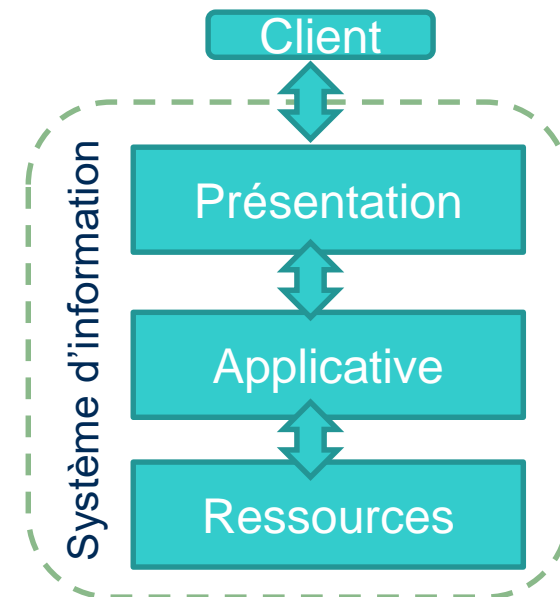
Introduction aux Architectures Distribuées



Les trois niveaux d'abstraction

Les niveaux d'abstraction

- En règle générale, une application est constituée de 3 niveaux d'abstraction:
 - La couche présentation (presentation layer)
 - La couche application (application logic layer)
 - La couche gestion des ressources (resource management layer)



La couche présentation (1/2)

- Tout système doit communiquer avec des entités externes:
 - Humains
 - Autres machines ...
- Besoin de présenter convenablement l'information à ces entités:
 - Objectif: permettre de soumettre des opérations et d'avoir les réponses
- Les éléments qui permettent ce type de traitement appartiennent à la couche présentation

La couche présentation (2/2)

- Les éléments de cette couche ne doivent pas être confondus avec le client.
- Le client est l'utilisateur du système
- Exemples de module de présentation:
 - Interface graphique (GUI)
 - Pages/Formulaires HTML
 - Application mobile

La couche application

- Le système doit délivrer de l'information
- Généralement, il effectue des tâches de calcul sur des données
- Il implémente les opérations requises par le client à travers la couche présentation
- Les modules, programmes et composants qui réalisent cette implémentation constituent la couche applicative.

La couche application

- Les éléments sont appelés: services
- Exemple: service de retrait sur compte bancaire:
 - Prend la requête de retrait
 - Vérifie que le compte est assez approvisionné
 - Vérifie que le plafond de retrait n'est pas atteint
 - Donne l'autorisation de retrait pour la somme demandée
 - Met à jour le nouveau solde
- Autre appellation pour ce niveau: processus business, logique business, règles business, ou simplement: serveur.

La couche gestion de ressources

- Les systèmes d'information ont besoin de données sur lesquels travailler
- Les données sont stockées sur
 - Fichiers
 - Bases de données
 - ERPs
 - ...
- La couche ressources correspond à ce type d'entités

Couche gestion de ressources

- Autre appellation: couche données
- Cela correspond au mécanismes de stockage persistant des données
- Dans des architectures plus complexes, la couche gestion de ressources peut aussi être un autre système d'information

Des couches conceptuelles vers les tiers

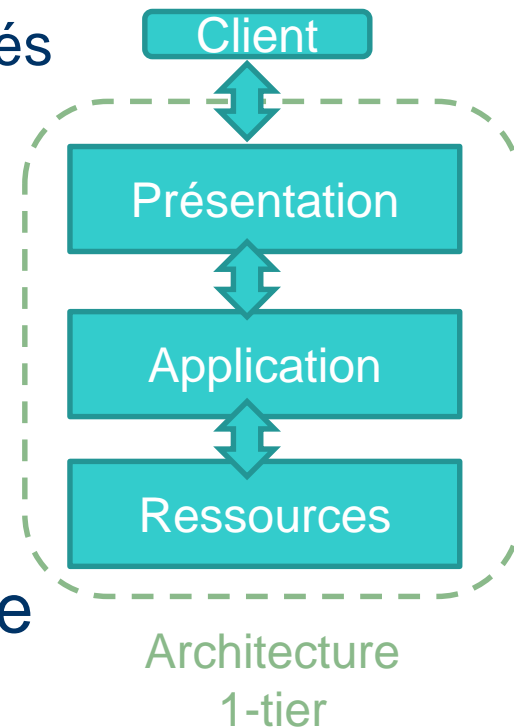
- Les trois couches présentation, application, et ressources sont des couches conceptuelles pour séparer les fonctionnalités d'un système
- Dans les systèmes réels, ces couches peuvent être distribuées et combinées de différentes manières
- On parle alors de tiers
- Selon l'organisation considérée pour ces tiers, on obtient les modèles architecturaux:
 - 1-tier
 - 2-tier
 - 3-tier
 - N-tier

L'architecture 1-tier



Il était une fois l'architecture 1-tier

- Historiquement, au début
 - Gros serveurs et calculateurs déconnectés
 - Une interface réduite à un invite de commande
 - Problématique principale: utiliser efficacement la CPU
- Systèmes monolithiques
- Les trois couches sont dans le même tier
- Le système vu comme une boîte noire
- Pas d'interaction avec d'autres systèmes ni d'API



Avantages

- Possibilité de fusionner à souhait les différentes couches pour optimiser l'application
- Pas besoin de maintenir et de publier une interface,
- Aucune raison d'investir dans des transformations complexes de données pour la compatibilité
- Coût nul concernant le développement des clients et le déploiement de l'application.
- Sécurisation et diagnostiquabilité plus simples

Inconvénients

- Un code monolithique et rigide
- Efficace mais très coûteux et difficile à maintenir
- Obsolète par rapport au matériel
- Absence des avantages liés à la distribution
- L'industrie du logiciel a pris le chemin opposé.



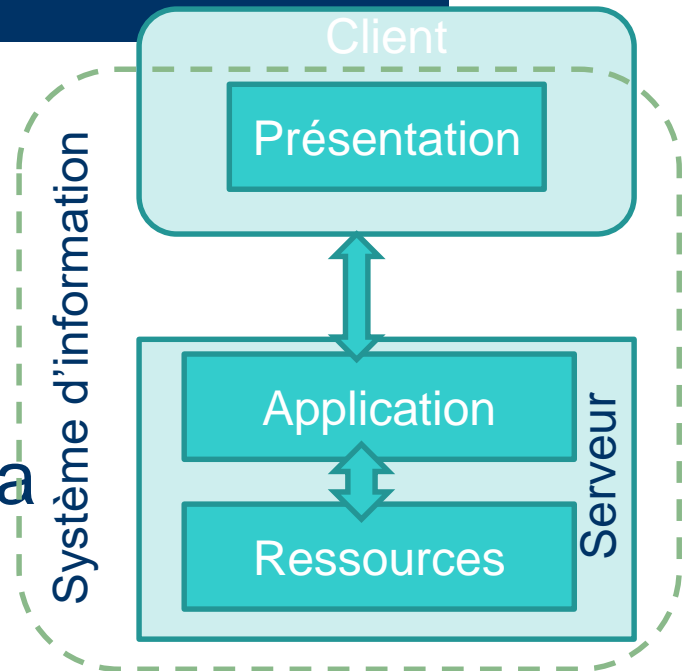
L'architecture 2- tier

L'architecture 2-tier

- Emergence due à l'apparition du PC
- Coexistence de machines moins puissantes (PC et stations de travail) avec de grosses machines (calculateurs et serveurs)
- Pour les designers:
 - Besoin de garder ensemble les couches gourmandes en ressources
 - La couche présentation est mise avec le client
- Avantages principaux:
 - Liés intrinsèquement à la distribution
 - Rendre possible la définition de plusieurs présentations pour la même application sans la rendre plus complexe
 - La couche présentation est déplacée dans le PC libérant de la puissance de calcul pour les deux autres couches

L'architecture 2-tier

- L'architecture 2-tier devient très populaire. On parle alors d'architecture Client-Serveur
- Le tier client correspond à la couche présentation
- Le tier serveur englobe les deux couches application et gestion des ressources



Développements associés à l'architecture Client-Serveur

- Les systèmes client-serveur ont permis plusieurs avancées dans les domaines du logiciel et du matériel avec une boucle vertueuse:
 - Avec l'augmentation des ressources dans les PCs et les stations de travail, la couche client est devenue de plus en plus sophistiquée.
 - La sophistication des clients a poussé vers une amélioration des performances dans les machines et dans les réseaux
- L'approche C-S est associée avec des développements cruciaux dans les systèmes distribués
 - La notion de RPC (Remote Procedure Call). Interaction à base d'appel de procédure.
 - Permet au concepteur de raisonner en terme d'interfaces publiées
 - La notion d'API.

Avantages sur le 1-tier

- La distribution
- Les couches application et ressources sont ensembles
 - L'exécution des opérations reste performante
- Indépendance du client et du serveur
 - Accès distants
 - Accès concurrents
 - Interopérabilité avec plusieurs plateformes
 - Possibilité de définir des couches présentation différentes pour différents clients à moindre coût

Inconvénients

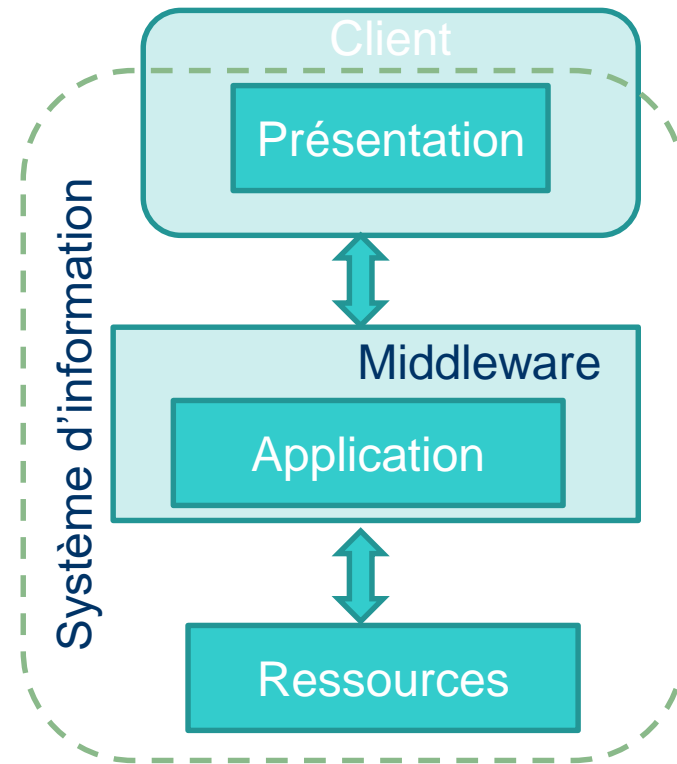
- Apparition de problématiques nouvelles
 - Les problématiques de communication
 - Gestion des accès concurrents
 - Passage à l'échelle (scalability)
 - Sécurité
 - La recherche et publication
 - Les transactions

Architecture 3-tier



L'architecture 3-tier

- Une séparation claire entre les différentes couches abstraites:
 - La couche présentation réside chez le client
 - La couche application réside dans le tier du milieu
 - L'infrastructure qui supporte le développement de la logique business est appelée middleware
 - La couche gestion de ressources réside dans un troisième tier



Comparaison avec l'architecture 2-tier

- Dans l'architecture 2-tier la couche application et gestion des ressources sont co-localisées
 - Avantage: le coût de la communication est nul
 - Inconvénient: il faut une machine puissante pour exécuter les deux
- Pour l'architecture 3-tier, séparation des deux couches
 - Avantage:
 - Possibilité de les distribuer sur différentes machines
 - Augmentation de la performance
 - Les deux sont moins liées
 - Reutilisabilité
 - Maintenabilité
 - Inconvénient:
 - Complexité de l'architecture
 - Coût de communication additionnel entre les deux couches

Développements liés à l'architecture 3-tier

- L'apparition d'architectures 3-tiers a engendré des avancées:
 - Les gestionnaires de ressources ont dû s'adapter pour offrir des interfaces permettant la communication avec la couche application qui s'exécute dans le middleware
 - Apparition de standards de communication pour les gestionnaires de ressources pour un accès uniforme de la couche application
 - Java DataBase Connectivity: JDBC
 - Object Relational Mapping; ORMs
- L'architecture 2-tier a forcé l'apparition d'API pour la couche application alors que l'architecture 3-tier a provoqué l'apparition d'API pour la couche gestion des ressources

Développements liés à l'utilisation des middlewares (1/2)

- L'apparition des architectures 3-tier a induit l'apparition de middleware permettant:
 - Une intégration aisée de la logique métier
 - Des fonctionnalités pour la gestion des ressources:
 - Recherche/Publication
 - Communication
 - Accès concurrents/gestion des instances
 - Persistence
 - Garanties transactionnelles
 - ...

Développements liés à l'utilisation des middlewares (2/2)

- Les concepteurs se concentrent sur la logique métier et profitent du support offert par le middleware pour le développement des interactions complexes
- La perte de performance liée à l'augmentation du coût de communication est généralement compensée par la possibilité de distribuer le tier du milieu sur plusieurs nœuds machine augmentant la scalabilité et la disponibilité de la communication



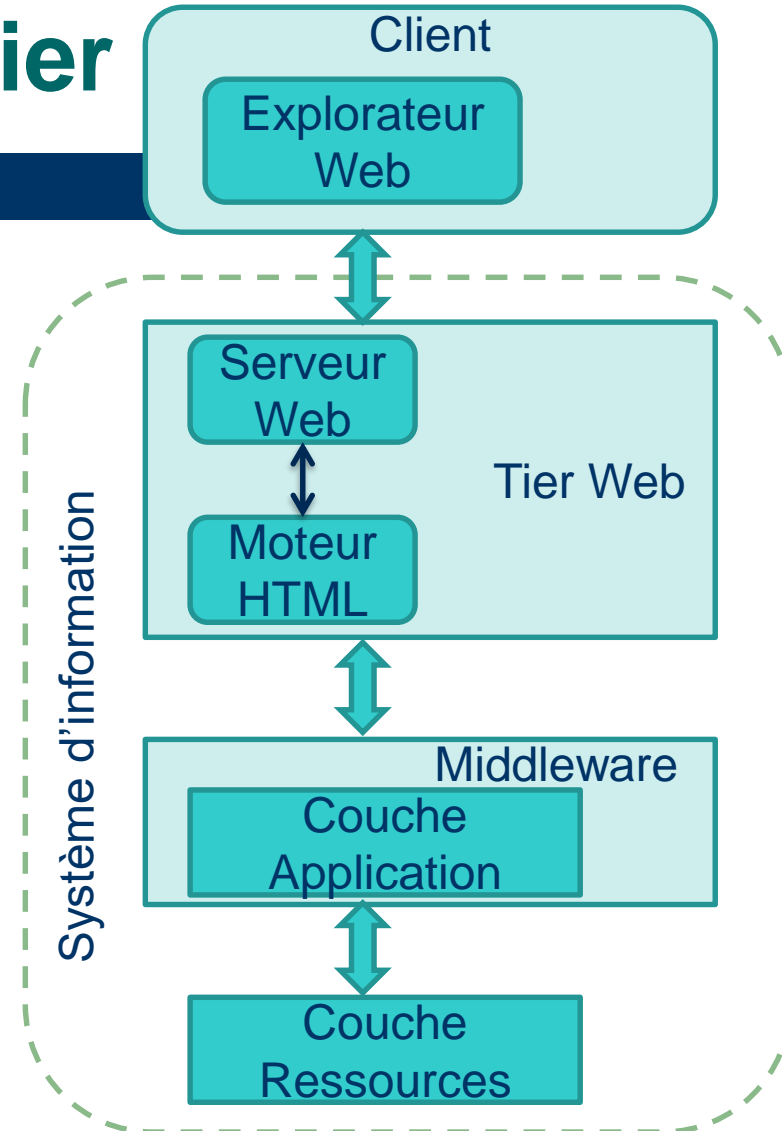
L'architecture N- tier

L'architecture N-tier

- Les architectures N-tier ne constituent pas réellement une évolution architectural par rapport à l'architecture 3-tier
- C'est une extension de ce modèle en considérant l'Internet comme un canal d'interaction
- La majorité des systèmes construits actuellement


Une architecture N-tier

- La couche présentation est scindée en deux tiers
 - Un tier client comprenant un explorateur Web,
 - Un tier Web comprenant le serveur Web et le code qui prépare les pages HTML
- Les tiers Application et Gestion de données gardent la même sémantique



La distribution

- En passant de l'architecture 1-tier vers la 2-tier, 3-tier, et N-tier, on assiste a une constante addition de tiers.
- Avec chaque tier,
 - l'architecture gagne en
 - Flexibilité
 - Fonctionnalité
 - Distribution
 - Introduit des coûts de communication additionnels entre les différents tiers
 - Introduit plus de complexité pour la gestion et la maintenance
- Il faut que le gain en flexibilité et en scalabilité compense les coûts de communication



Communication dans les systèmes d'information

La communication dans les systèmes d'information

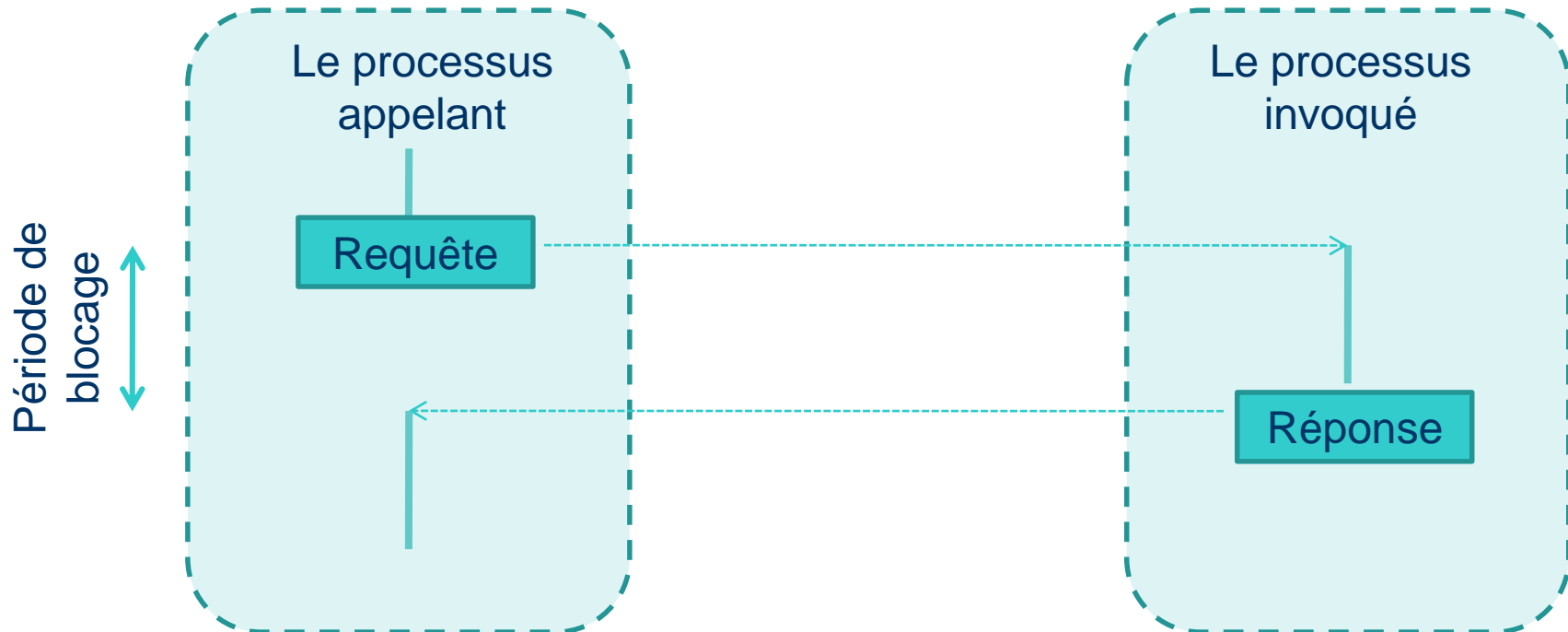
- Nous avons discuté comment les couches abstraites et les tiers peuvent être combinés et distribués
- Séparer le système en plusieurs tiers implique l'implémentation des mécanismes de communication entre ces éléments
- La caractéristique dominante des différents modes d'interaction correspond aux choix synchrone ou asynchrone



Interactions Synchrones

Interaction synchrone

- Dans une interaction synchrone un processus qui appelle un autre doit attendre la réponse avant de continuer ses traitements.



Avantages (1/2)

- Attendre la réponse avant de continuer présente plusieurs avantages:
 - Simplifier la conception
 - L'état du processus appelant n'est pas altéré entre son appel et la réponse
 - Corrélation simple entre le code qui fait l'appel et le code qui traite la réponse (les deux bouts de code sont en séquence)
 - Les composants sont fortement liés pour chaque interaction. Plus de facilité pour le test, le débogage, et l'analyse de performance

Avantages (2/2)

- Au passage de l'architecture 1-tier vers 2-tier, la majorité des systèmes utilisent du synchrone pour la communication entre clients et serveur
- Au passage vers l'architecture 3-tier, la majorité des serveurs de données offrent une communication synchrone avec la couche application

Inconvénients

- Tous les avantages peuvent être aussi vus comme des inconvénients quand l'interaction n'est pas de type requête-réponse (e.g. one way)
- Perte de temps et de ressources calcul si le traitement côté serveur est long
- Le problème de performance augmente avec l'augmentation du nombre de tiers
- En terme de tolérance aux fautes, le processus appelant et le processus invoqué doivent être connectés et opérationnels lors de l'invocation. Ils doivent le rester tout au long de l'exécution de la requête
- Les procédures de maintenance doivent être faites offline



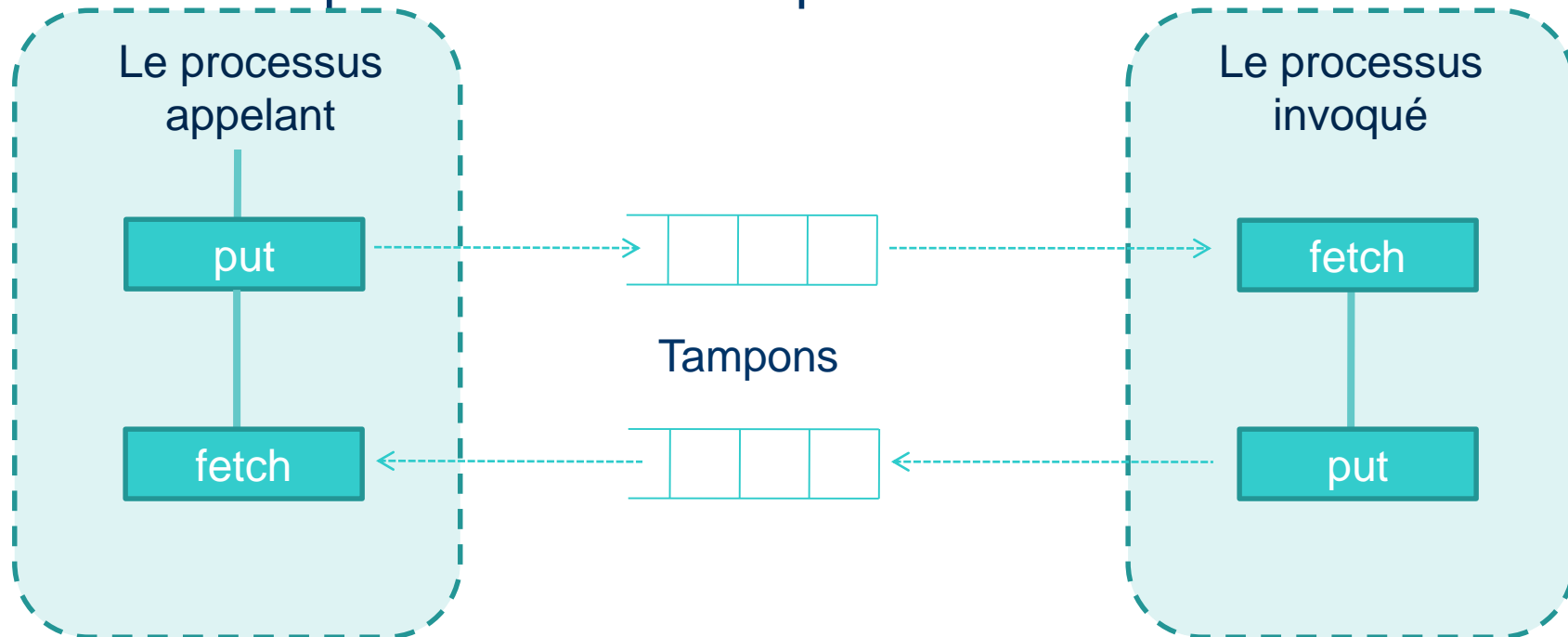
Interactions asynchrones

Les interactions asynchrones

- Quand il est nécessaire de travailler de manière interactive, le choix synchrone s'impose
- Dans plusieurs cas, cela n'est pas nécessaire
 - Impression
 - J'envois une demande d'impression
 - La demande est insérée dans la liste des tâches
 - L'imprimante traite la tâche quand elle est disponible
 - La machine est notifiée de la fin du traitement

Les interactions asynchrones

- Au lieu de faire une requête et attendre la réponse
 - Envoyer la requête
 - Vérifier plus tard si une réponse a été retournée



Avantages

- Suivre une approche non bloquante permet:
 - Au programme appelant de continuer à réaliser d'autres tâches pendant le traitement de sa requête
 - D'éliminer le traitement de la coordination entre les deux processus
- Réduction des problèmes dus
 - Au nombre de connections
 - À la dépendance entre composants
 - À la tolérance aux fautes
- Modèle adéquat dans certaines situations (style publisher-subscriber)
 - Un serveur dissémine l'information vers plusieurs clients
 - Différents clients s'intéressent à différents types d'information



Evolution des concepts de développement logiciel

Applications Procédurales

- E.g. C, Pascal, Ada
- L'univers est constitué principalement de procédures et de fonctions
- Interactions principalement via une interface IHM locale

L'orienté objet

- E.g. Java, C++, Eiffel
- L'univers est constitué principalement d'objets
 - Attributs
 - Méthodes
- L'orienté objet est plus une vision relative à l'organisation de l'implantation
 - La distribution et le déploiement ne sont pas pris en compte
- Concepts nouveaux:
 - Attributs (caractéristiques d'un objets)
 - Méthodes (actions possibles sur un objet)

L'orienté composant

- E.g. RMI, CORBA, EJB
- On s'intéresse maintenant à la structure en terme de
 - Décomposition du code exécutable stand-alone (composants)
 - Déploiement au niveau des environnements d'exécution
- Objectifs principaux, au sein d'une entité:
 - Réutilisation
 - Composition
 - Facilitation de développement
- Nouveaux Concepts:
 - Interfaces
 - Connexions
 - Middleware

L'orienté service

- E.g. Services Web
- Au-delà de l'orienté-composant
- Contexte: le web et les applications inter-entités
- Objectifs principaux
 - Développer le e-business
 - Publier-rechercher des services sur le web
 - Composition dynamique de services
- Nouveaux concepts
 - Registres de publications
 - Standards de
 - Description
 - Communication
 - Publication

Paradigmes de communication

- MOM
 - Orienté messages
- RPC
 - Orienté procédures
- RMI
 - Orienté méthodes



Ce qu'il faut retenir

Retour sur les thèmes abordés

- Trois couches conceptuelles sont identifiées
 - Présentation
 - Application
 - Gestion de ressources
- Quatre modèles architecturaux
 - 1-tier, 2-tier, 3-tier, N-tier
- Deux modèles de communication:
 - Client-Serveur
 - Publisher-Subscriber

La distribution, c'est Darwin!

- La distribution des couches conceptuelles a évolué en réponse à des évolutions au niveau matériel et réseau
 - Avec les gros serveur et calculateurs: l'architecture 1-tier
 - Avec les réseaux locaux et l'apparition des PCs et des stations de travail: l'architecture client-serveur
 - Avec la prolifération de l'information et l'augmentation de la bande passante: l'architecture 3-tier
 - Avec l'avènement d'Internet, une bande passante en augmentation constante, et l'essor du e-business: l'architecture N-tier

L'évolution de la programmation

- De programmes monolithique isolés
- Vers une structuration des applications en services distribués à travers le Web
- Développement d'outils et de standards
 - Conception
 - Développement
 - Déploiement
- De plus en plus de génération automatique de code
 - Stubs / Skeletons, classes de support
- Décharger un maximum le développeur pour se consacrer à la logique business



Introduction à la plateforme JEE

Le cadre (framework) java 2

- Le framework java comporte 3 éditions:
 - JSE: Java Standard Edition destinée au développement d'applications locales exécutables sur des ordinateurs personnels
 - JEE: Java Enterprise Edition destinée au développement d'applications distribuées impliquant des serveurs
 - JME: Java Micro Edition destinée au développement d'applications embarquées pour les terminaux mobiles.

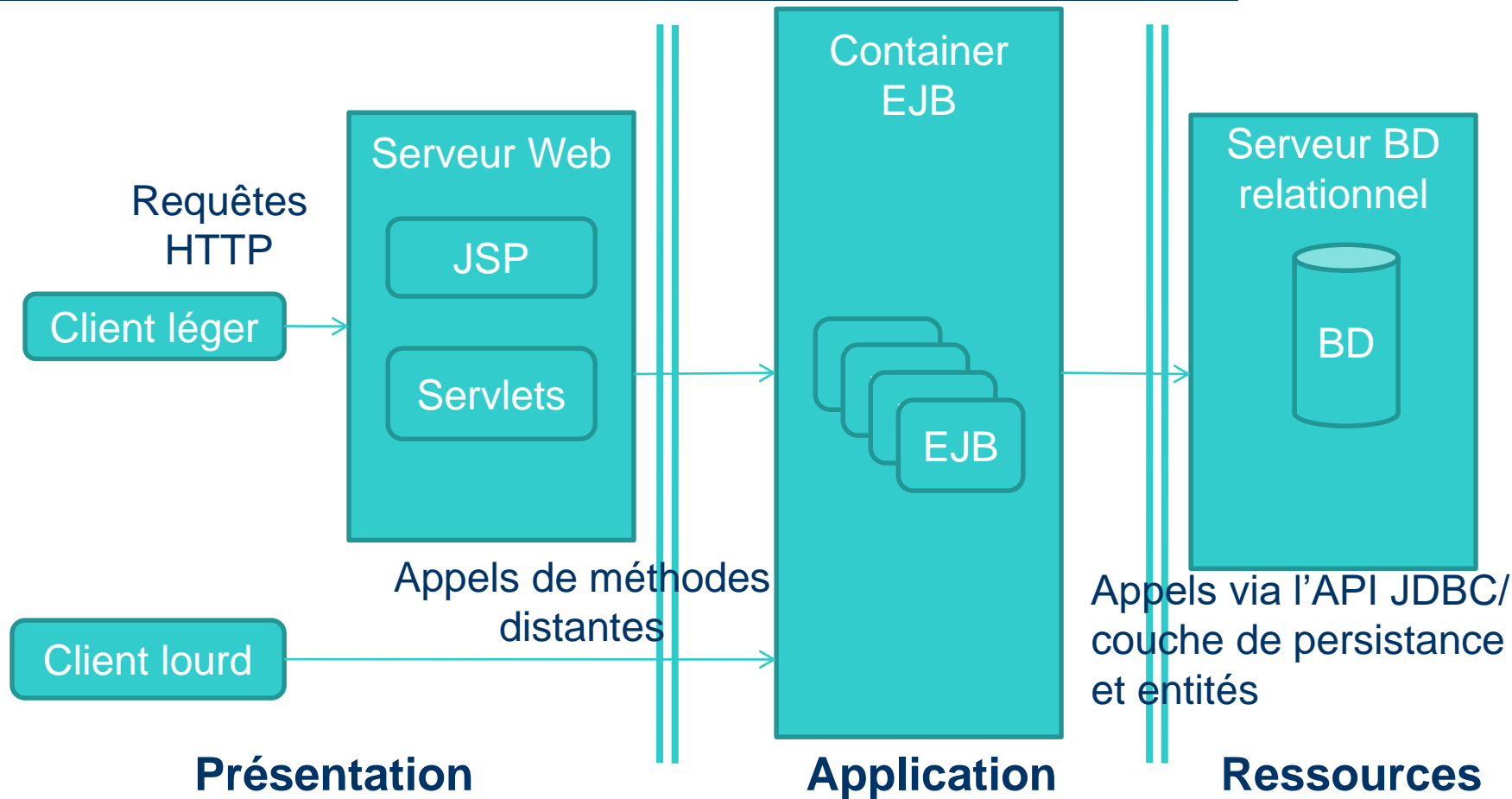
La plateforme JEE

- La plateforme JEE propose une approche **orientée-composants** pour la conception, le développement, le déploiement des applications distribuées
- JEE:
 - Repose sur un **modèle architectural N-tier**
 - Favorise la réutilisabilité des composants logiciels
 - Propose un modèle de gestion de la sécurité et des transactions
- Versions:
 - J2EE 1.2 (1999) / J2EE 1.3 (2001) / J2EE 1.4 (2003) / Java EE 5(2006) / Java EE 6 (2009) / Java EE 7 (2013) / Java EE 8 (2017) / Jakarta EE 8 (2019)

L'architecture JEE

- Une architecture JEE classique comprend les tiers suivants:
 - Le tier Client
 - Le tier Web
 - Le tier logique business
 - Le tier systèmes d'information

Exemple d'une architecture JEE



Les composants JEE (1/2)

- Les composants d'une architecture correspondent à des unités software "self-contained" possédant une interface définie
- Les applications JEE sont construites par assemblage de composants JEE
- La spécification JEE définit les composants suivants:
 - Les composants clients: clients applicatifs, clients web
 - Les composants web: Servlets et JSP
 - Les composants métier: EJB

Les composants JEE (2/2)

- Les composants JEE sont
 - Développés en langage java
 - Compilés comme tout programma java
- A la différence d'une classe JSE
 - Les composants sont connectés pour former l'application
 - Leurs implémentation et leurs interfaces sont vérifiées pour s'assurer de leur conformité à la spécification JEE
 - Les exécutables sont déployés sur le serveur JEE pour la production

Les clients JEE



Les clients applicatifs

- S'exécutent sur la machine du client
- Généralement, s'adressent directement au tier métier (e.g. des appels distants sur un composant EJB)
- Peuvent correspondre à
 - Une interface graphique Swing ou AWT
 - Une interface en ligne de commande
 - Une application mobile
 - Une autre application

Les clients Web

- Un client Web est constitué de deux parties
 - Les pages web dynamiques de type HTML ou XML générées par les composants web
 - Un browser qui interprète ces pages
- Généralement, un client Web est un client léger avec très peu de traitements

Les composants web



Les servlets

- Une servlet est un programme java déployé sur un serveur web
- Chargée automatiquement dans le serveur ou à la demande d'un client
- Une fois déployée elle reste en attente des requêtes clients
- Génère dynamiquement des données sous format de pages Web (format HTML ou XML)

Les JSPs

- Composants permettant de générer dynamiquement des pages web
- Le modèle JSP est dérivé du modèle servlet
- À l'appel d'un client, le serveur Web appelle le moteur JSP pour générer le code source, le compile pour générer l'exécutable de la servlet qui traite la requête



Les composants Java Bean

Les EJB

- La partie métier est implantée sous forme de composants EJB
- Les EJB réalisent les traitements, ils constituent le tier application
- Généralement, un EJB
 - reçoit l'information du tier client la traite et met à jour les données stockées dans le tier gestion des ressources
 - récupère l'information stockée dans le tier gestion des ressources la traite et renvoie le résultat de ses traitements au client

Les EJB

- Trois types d'EJB peuvent être utilisés:
 - Les session bean: non persistents
 - Les entity bean: persistents
 - Les messages-driven bean: combinaison de session bean et de JMS



Les composants gestion de ressources

La gestion de ressources

- Les composants correspondent à des entités logicielles mettant des données persistantes à la disposition des composants métier
- Les ressources traitées par une application JEE peuvent correspondre à:
 - une base de données relationnelle
 - un Entreprise Ressource Planning (ERP)
 - un système de fichiers
- Une couche de persistance peut être considérée afin de permettre
 - d’avoir une représentation objet de la BD
 - de gérer les aspects transactionnels et d’accès concurrents

Le middleware JEE



Les containers

- Un container constitue l'interface entre un composant et les fonctionnalités de bas niveau qu'offre la plateforme JEE tels que:
 - Le service JNDI: service de nommage
 - Le modèle security: gestion des accès et des authentications
 - Le modèle transactionnel: gestion des transactions pour les accès aux données
- Avant l'exécution d'une application, ses composants doivent être assemblés en modules JEE et déployés dans un container

Les containers JEE

- Le container EJB
 - Gère l'exécution des EJBs.
 - Les EJBs et leurs containers s'exécutent au sein du serveur JEE
 - La majorité des serveurs JEE offrent des containers EJB intégrés
- Le container Web
 - Gère l'exécution des servlets et des JSPs
 - Les composants Web et leurs containers s'exécutent au sein du serveur JEE

Le serveur JEE

- Se charge de l'exécution des composants JEE
- Offre des containers Web et EJB
- Les serveurs JEE
 - Glassfish (Oracle)
 - JBOSS (Red Hat)
 - WebSphere Application Server (IBM)
 - Geronimo (Apache)



Présentation de l'API JEE

Quelques éléments de l'API

- Les EJBs (Enterprise Java Beans)
 - Composants métiers
 - Package: javax.ejb
- Les servlets
 - Composants Web
 - Package: javax.servlet
- Les JSP (Java Server Pages)
 - Composant Web
 - Package: javax.servlet.jsp

Quelques éléments de l'API

- JMS (Java Messaging Service)
 - Permet des échanges de messages asynchrones entre les composants
 - Package: javax.jms
- JavaMail
 - Permet l'envoi des notifications mail
 - Package: javax.mail
- JAXP (Java Api for Xml Processing)
 - Permet l'analyse et le parsing des fichiers XML
 - Package: javax.xml.parsers

Quelques éléments de l'API

- JAX-RPC (Java API for XML-Based RPC)
 - Permet des interactions RPC à base de messages SOAP
 - Package: javax.xml.rpc
- JTA (Java transaction API)
 - Permet la gestion des transactions
 - Package: javax.transaction
- JNDI
 - Implémente de service de nommage dans JEE
 - Package: javax.naming



Les Servlets

C'est quoi une servlet?

- Une servlet est un composant Web
- Déployée du côté serveur, sur un serveur Web
 - Reçoit une requête du client Web
 - Effectue des traitements
 - Renvoie une réponse

Fonction

- L'utilité première d'une servlet est la génération dynamique de pages html
- Généralement, les servlets utilisent le protocole http pour communiquer
 - Requêtes http
 - Réponses http
- Cependant, une servlet peut reposer sur n'importe quel protocole bâti selon le modèle requête/réponse

Avantages de l'utilisation des servlets

- La génération **dynamique** des pages html
- Les servlets sont codées en java
 - Portabilité
 - Accès aux APIs Java
 - Possibilité d'interaction avec
 - Les composants métier
 - Les composants gestion des ressources (BD, ERP, ...)
 - Garbage collector

Fonctionnement d'une servlet

- La servlet construit dynamiquement la page HTML réponse et l'encapsule dans l'objet réponse transmis par le serveur en utilisant
 - Les valeurs transmises par la requête
 - Toute autre ressource disponible et accessible
 - Les données persistantes
 - Les composants métier
 - ...
- Le serveur récupère la page HTML contenue dans l'objet réponse de la servlet
- Le transmet au client (browser)

Fonctionnement d'une servlet

- La servlet est déployée
 - En copiant son code exécutable dans un répertoire spécifique du serveur Web (pour Tomcat, c'est le répertoire webapps)
 - En définissant son descripteur de déploiement (pour Tomcat, web.xml)
- Le serveur Web charge la servlet automatiquement ou à la demande d'un client
- Une servlet offre un accès concurrent

Fonctionnement d'une servlet utilisant le protocole HTTP

- Le serveur reçoit une requête d'un navigateur spécifiant l'appel à une servlet
 - lien http correspondant à l'url définie pour la servlet dans le descripteur de déploiement
- A la première requête d'un client, le serveur instancie la servlet
- Si elle n'est pas arrêtée de manière active, la servlet reste en mémoire jusqu'à l'arrêt du serveur

Fonctionnement d'une servlet utilisant le protocole HTTP

- Pour chaque requête, le serveur crée un processus léger pour son traitement
 - Possibilité de traiter plusieurs requêtes à la fois
 - Limite spécifiée dans la configuration du serveur
 - Pour Tomcat, 150 par défaut (c.f. fichier XML conf/server.xml)
 - Le serveur crée
 - Un objet représentant la requête HTTP
 - Un objet qui va contenir la réponse HTTP
 - Envoie les deux objets à la servlet
 - Renvoie le résultat encapsulé dans la réponse au client

Les serveurs Web

Tomcat (apache)



Rôle du serveur Web

- Servir de plateforme d'exécution des servlets
- Gestion du cycle de vie
 - Chargement/rechargement
 - Démarrage/arrêt/retrait
 - Instanciation
- Gestion des interactions
 - Interceptions des requêtes destinées aux servlets
 - Création des objets requêtes et réponse
 - Retour des pages HTML résultat

Le serveur Web Tomcat

- Développé par la fondation Apache
- Implémente les spécifications servlet et jsp
- Utilise le container catalina (pour servlets)
- Compile les JSPs grâce au compilateur jasper
- Coyote est le connecteur http de tomcat
- Codé en Java
- Dernière version: 9.0.26 (Septembre 2019)

Installation de Tomcat

- Récupérer le code de Tomcat
- Le copier dans un répertoire (e.g. TomcatRep)
- Positionner correctement les variables d'environnements:
 - TOMCAT_HOME à TomcatRep
 - CATALINA_HOME à TomcatRep
 - JAVA_HOME au répertoire du JDK
- Pour pouvoir lancer Tomcat à partir d'un invite de commande mettre à jour la variable PATH: rajouter le chemin, TomcatRep\bin

Le contenu du code

- bin : Scripts et exécutables pour différentes tâches : démarrage, arrêt,
- lib : Classes communes que Catalina et les applications Web utilisent
- conf : Fichiers de configuration (format XML)
- logs : Journaux des applications Web et de Catalina ;
- webapps : Répertoire contenant les applications web déployées sur Tomcat
- work : Fichiers et répertoires temporaires.

Lancement de Tomcat

- Si le PATH mis à jour,
 - Ouvrir un invite de commande et lancer le programme via la commande: startup
- Sinon, aller sur le répertoire TomcatRep/bin et lancer l'exécutable startup
- Pour vérifier si Tomcat est correctement lancé,
 - Aller sur l'url: <http://localhost:8080>

Le manager de Tomcat

- url: <http://localhost:8080/manager>
- Permet de gérer le cycle de vie des composants Web
- Par défaut, il faut positionner un administrateur avec le rôle manager
- Aller sur le fichier TomcatRep/conf/tomcat-users.xml
- Rajouter un utilisateur avec une balise de la forme
 - `<role rolename="manager-gui"/>`
 - `<user username="karim" password="karim" roles="manager-gui"/>`

L'API Servlet

Les packages

- Deux packages
 - javax.servlet:
 - Permet de développer des servlets génériques indépendantes du protocole d'interaction
 - javax.servlet.http
 - Permet de développer des servlets interagissant avec le protocole HTTP



Le package javax.servlet

Les interfaces

- Servlet: définition de base d'une servlet
- ServletConfig: définition d'un objet pour configurer la servlet
- ServletContext: définit un objet contenant des informations sur le contexte d'exécution de la servlet
- ServletRequest: définit un objet contenant la requête du client
- ServletResponse: définit un objet contenant la réponse

Les classes

- `GenericServlet`: définit une servlet indépendante de tout protocole
- `ServletInputStream`: définit un flux permettant la lecture des données de la requête cliente
- `ServletOutputStream`: définit un flux permettant le renvoi de la réponse de la servlet

Les exceptions

- `ServletException`: Exception générale levée en cas de problème durant l'exécution de la servlet
- `UnavailableException`: Exception levée si la servlet n'est pas disponible

L'interface Servlet

- `void init(ServletConfig conf):`
 - Appelée une seule fois lors de l'instanciation de la servlet
- `ServletConfig getServletConfig():`
 - Renvoie l'objet `ServletConfig` passé à la méthode `init`
- `void service (ServletRequest req,ServletResponse res):`
 - A chaque requête du client cette méthode est exécutée
- `void destroy():`
 - Permet la destruction de la servlet
- `String getServletInfo():`
 - Renvoie les informations définies pour la servlet.

L'interface ServletRequest

- Méthodes
 - ServletInputStream getInputStream()
 - BufferedReader getReader()
 - Permettent d'obtenir un flux de lecture sur les données de la requête

L'interface ServletResponse

- Méthodes
 - SetContentType: Permet de préciser le type de la réponse
 - /text/html pour une page html
 - ServletOutputStream getOutputStream()
 - Permet d'obtenir un flux pour envoyer la réponse
 - PrintWriter getWriter()
 - Permet d'obtenir un flux d'écriture pour envoyer la réponse



Démonstration d'implémentation



Implémentation d'une servlet

- Une servlet est codée comme n'importe quelle classe java
- Implémente l'interface `javax.servlet.Servlet`
- Définir, entre autres, la méthode
 - `service (ServletRequest req, ServletResponse res)` permettant de récupérer la requête et de retourner le résultat

Une servlet qui dit bonjour

- Dans un fichier MaPremiereServlet.java:

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
public class MaPremiereServlet implements  
    Servlet {
```

```
    private ServletConfig cfg;
```

Implémentation des méthodes de l'interface Servlet

```
public void init(ServletConfig config) throws ServletException  
{cfg = config;}
```

```
public ServletConfig getServletConfig()  
{ return cfg; }
```

```
public String getServletInfo()  
{return("C'est ma première Servlet");}
```

```
public void destroy() { }
```

La méthode service

```
public void service (ServletRequest req, ServletResponse res )
throws ServletException, IOException {
res.setContentType( "text/html" );
PrintWriter out = res.getWriter();
out.println( "<HTML>" );
out.println( "<HEAD>");
out.println( "</HEAD>" );
out.println( "<BODY>" );
out.println( "<H1 align=\"center\">Je suis content ma premiere Servlet me
    repond:</H1>" );
out.println( "<H2>Bonjour</H2>" );
out.println( "</BODY>" );
out.println( "</HTML>" );
out.close();}}
```

Compilation de la servlet

1. S'assurer que les variables d'environnement relatifs à java sont bien positionnés
 1. JAVA_HOME=...
 2. JDK_HOME=...
 3. PATH=%PATH%;...\TomcatRep\bin
2. S'assurer que le package JEE est inclus dans le classpath
 1. classpath=.;C:\java\lib\tools.jar;C:\java\jre\lib\rt.jar
 2. classpath=%classpath%;java\lib\Javaee.jar
3. Compiler la classe MaPremiereServlet
 1. javac MaPremièreServlet.java

Arborescence du code exécutable

- Sous le répertoire **webapps** de Tomcat
 - Un répertoire correspondant à l'application Web, sous lequel se trouvent:
 - Un répertoire **WEB-INF** qui contient au moins:
 - Un répertoire **classes**
 - Les exécutables
 - Un fichier **web.xml**
 - Le descripteur de déploiement

Descripteur de déploiement

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <servlet>
    <servlet-name>Toto</servlet-name>
    <servlet-class>HelloWorldExample</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Toto</servlet-name>
    <url-pattern>/Toto</url-pattern>
  </servlet-mapping>
</web-app>
```

Exécution

- Lancer Tomcat
- Ouvrir le browser
- Envoyer une requête à la servlet sur l'url:
 - `http://localhost:8080/NomApplicationWeb/URL_Servlet`

Les servlets HTTP

Le package



La classe HttpServlet

- L'API fournit une classe qui encapsule une servlet utilisant le protocole http
- La classe HttpServlet
 - Hérite de la classe GenericServlet qui implémente l'interface Servlet
 - Redéfinie toutes les méthodes nécessaires pour fournir un niveau d'abstraction permettant de développer facilement des servlets avec le protocole http

Interaction avec les servlets http

- La méthode service héritée de HttpServlet appelle l'une ou l'autre de ces méthodes en fonction du type de la requête http :
 - une requête GET : c'est une requête qui permet au client de demander une ressource
 - une requête POST : c'est une requête qui permet au client d'envoyer des informations issues par exemple d'un formulaire
 - Des requêtes PUT, DELETE...
- En pratique, la méthode service analyse HttpServletRequest et exécute doGet, doPost, doPut... selon le type de la requête

Le package `javax.servlet.http`

- Les interfaces
 - `HttpServletRequest`: Hérite de `ServletRequest` et définit un objet contenant une requête selon le protocole http
 - `HttpServletResponse`: Hérite de `ServletResponse` et définit un objet contenant la réponse de la servlet
 - `HttpSession`: Définit un objet qui représente une session d'interaction avec la servlet

Démonstration d'implémentation

Implémentation d'une servlet http

- Hérite de la classe HttpServlet
- Définir les méthodes
 - doGet(HttpServletRequest req, HttpServletResponse res) permettant de renvoyer la réponse pour une requête http Get
 - doPost(HttpServletRequest req, HttpServletResponse res) permettant de renvoyer la réponse pour une requête http Post

Une servlet http qui dit bonjour

- Dans un fichier MaDeuxiemeServlet.java:

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
public class MaDeuxiemeServlet extends  
    HttpServlet {
```

Implémentation de la méthode Get

```
public void doGet (HttpServletRequest req, HttpServletResponse res )
throws ServletException, IOException {
res.setContentType( "text/html" );
PrintWriter out = res.getWriter();
out.println( "<HTML>" );
out.println( "<HEAD>" );
out.println( "</HEAD>" );
out.println( "<BODY>" );
out.println( "<H1 align=\"center\">Je suis content, ma Servlet HTTP me répond</H1>" );
out.println( "<H2>Bonjour</H2>" );
out.println( "</BODY>" );
out.println( "</HTML>" );
out.close();
}
```

Implémentation de la méthode Post

```
public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
{
    this.doGet(request, response);
}
}
```


Déploiement de la servlet sur Tomcat5.5

- Déployer la servlet
 - Copier le fichier MaDeuxiemeServlet.class dans le répertoire
 - CheminTomcat\webapps\...\WEB-INF\classes
 - Modifier le descripteur de déploiement, fichier :
 - CheminTomcat\webapps\...\WEB-INF\web.xml

Modification du descripteur web.xml

- Rajouter les balises

- `<servlet>`

- `<servlet-name>MaDeuxiemeServlet</servlet-name>`

- `<servlet-class>MaDeuxiemeServlet</servlet-class>`

- `</servlet>`

- `<servlet-mapping>`

- `<servlet-name>MaDeuxiemeServlet</servlet-name>`

- `<url-pattern>/MaDeuxiemeServlet</url-pattern>`

- `</servlet-mapping>`



Utilisation des formulaire

Une servlet http qui dit bonjour en réponse à un formulaire

- Dans un fichier MaTroisiemeServlet.java:

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
public class MaTroisiemeServlet extends  
    HttpServlet {
```

Implémentation de la méthode Get

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response)
    throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Bonjour tout le monde</title>");
    out.println("</head>");
```

Implémentation de la méthode Get

```
...
out.println("<body>");
out.println("<FORM
    ACTION=\"http://localhost:8080/UneAppli/MaTroisiemeServlet\"
    METHOD=\"POST\">");
out.println("<H3> Votre nom s il vous plaît:");
out.println("<INPUT NAME=\"NOM\">");
out.println("<INPUT TYPE=\"SUBMIT\" Value=\"envoyer\">");
out.println("</FORM>");
out.println("</body>");
out.println("</html>");
}
```

Implémentation de la méthode Post

```
public void doPost(HttpServletRequest request, HttpServletResponse  
    response)  
    throws ServletException, IOException {  
    response.setContentType( "text/html" );  
    PrintWriter out = response.getWriter();  
    out.println( "<HTML>" );  
    out.println( "<HEAD>" );  
    out.println( "</HEAD>" );  
}
```

Implémentation de la méthode Post

```
...
out.println( "<BODY>" );
out.println( "<H1 align=\"center\">Je suis content ma Troisième
    Servlet me répond personnellement!</H1>" );
out.println( "<H2>Bonjour "+request.getParameter("NOM")+"</H2>" );
out.println( "</BODY>" );
out.println( "</HTML>" );
out.close();
}
}
```


Déploiement de la servlet sur Tomcat5.5

- Déployer la servlet
 - Copier le fichier MaTroisiemeServlet.class dans le répertoire
 - CheminTomcat\webapps\examples\WEB-INF\classes
 - Modifier le descripteur de déploiement, fichier :
 - CheminTomcat\webapps\examples\WEB-INF\web.xml

Modification du descripteur

- Rajouter l'élément

- `<servlet>`

- `<servlet-name>MaTroisiemeServlet</servlet-name>`

- `<servlet-class>MaTroisiemeServlet</servlet-class>`

- `</servlet>`

- `<servlet-mapping>`

- `<servlet-name>MaTroisiemeServlet</servlet-name>`

- `<url-pattern>/MaTroisiemeServlet</url-pattern>`

- `</servlet-mapping>`

Un petit exercice:

- Implémenter deux servlets
 - Une servlet Ping
 - Avec un bouton ping pour le doGet
 - Une servlet Pong
 - Avec un bouton pong pour le doGet
 - Le bouton ping de la servlet Ping renvoie vers la servlet Pong
 - Le bouton pong de la servlet Pong renvoie vers la servlet Ping
- Utiliser des servlets sur les différentes machines en fournissant les adresses IP des collègues

Un deuxième exo

- Un servlet Accueil permettant de choisir une opération de
 - Soustraction
 - Ou d'addition
- Une servlet Addition permettant de
 - Saisir deux valeurs entières
 - D'afficher leur somme
 - Et de fournir un bouton retour
- Une Servlet Soustraction avec un fonctionnement similaire à la servlet Addition



Les Java Server Pages(JSP)



La technologie JSP

- Les JSP constituent une technologie Java dont l'objectif est de permettre la génération dynamique des pages web
- L'atout majeur est de permettre la séparation des traitements et de la présentation au niveau du code de développement
- Généralement,
 - Présentation sous forme de code HTML
 - Traitement sous forme de code Java insérée dans des tags spéciaux

Fonctionnement

- Les JSPs constituent « un script » pour le développement de servlets
- Au premier appel de la JSP, le serveur web
 - Génère la servlet correspondante
 - Le code HTML est repris dans la servlet
 - Le code Java est inséré dans le code de la servlet
 - Compile la servlet
 - Charge la servlet et l'exécute
- Pour les appels suivants, la servlet est directement exécutée, donc plus rapide

JSP Vs Servlet

- JSPs et Servlets possèdent beaucoup de points communs (une JSP est traduite en une servlet!)
 - Les JSPs permettent un développement moins pénible (pas besoin de flux pour la génération de la page html) pour la partie présentation
 - Pas de phase de compilation active
 - Facilite la mise à jour et la modification
 - Pas de descripteur de déploiement
- Code HTML versus Code Java
 - Code HTML : JSP
 - Code Java: Servlet

Contenu d'une JSP

- Une JSP est un fichier qui possède, par convention, l'extension jsp
- Un fichier JSP est constitué de:
 - de tags HTML
 - de tags JSP
 - Directives
 - Scripting



Développons une JSP: La démo d'illustration

Une JSP correspondant à une page HTML (uniquement des tags HTML)

- Ecrivons une JSP renvoyons le même code de la première servlet
- Dans un fichier JSP_statique.jsp introduire le code HTML suivant:

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<H1 align="center">On est tous contents</H1>  
<H2>Nous ecrivons notre premiere JSP</H2>  
</BODY>  
</HTML>
```

Déploiement

- Enregistrer le fichier jsp sous webapps, par exemple sous:
 - /MesPremieresJSPs/JSP1.jsp
- Sur le navigateur accéder à la jsp par l'url:
 - <http://localhost:8080/MesPremieresJSPs/JSP1.jsp>

Regardons ce qui se passe au niveau de Tomcat

- Le code d'interprétation du script JSP vers le servlet est généré dans le répertoire `/tomcat/work/.../MesPremieresJSPs`
- L'exécutable est aussi généré dans ce répertoire
- Modifier le code de la JSP (e.g. supprimer une ligne)
- Recharger la jsp
- Revérifier le code de la servlet générée
- Appeler la servlet
- Re-revérifier le code



Plus loin



Les tags jsp

- En plus des tags HTML, il existe deux types de tags jsp :
 - Les tags de directives : permettent de contrôler la structure de la servlet générée
 - Les tags de scripting: permettent d'insérer du code Java dans la servlet

Les tags de directives

- Les tags de directives possèdent la syntaxe:
 - `<%@ directive attribut="valeur" ... %>`
- Les spécifications des JSP les directives :
 - `page` : permet de définir des options de configuration
 - `include` : permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet

La directive page (1/2)

- Possède les attributs:
 - `contentType="text/html;charset=UTF-8"`
 - permet de préciser le type des données générées.
 - équivalente à `response.setContentType(..);`
 - `isErrorPage="true|false"`
 - Cette option permet de préciser si la JSP génère une page d'erreur.
 - `errorPage="URL"`
 - Cette option permet de préciser la JSP appelée au cas où une exception est levée

La directive page (2/2)

- extends="classe mere"
 - Permet de spécifier une classe mère pour la servlet générée
- import= "{package.class}":
 - Permet de spécifier les packages à importer pour le code java inclus dans la jsp
- info="Les infos":
 - permet de spécifier la chaîne de caractères renvoyée par la méthode `getServletInfo`
- isThreadSafe="true|false":
 - permet de spécifier au serveur web le mode d'interaction de la servlet générée (un seul traitement de requête à la fois (safe=false) ou plusieurs en parallèle(safe=true))
- ...

La directive include

- Syntaxe:
 - `<%@ include file="chemin du fichier" %>`
- Permet d'inclure un fichier dans le code source de la jsp
- Ce fichier peut correspondre à
 - Du code HTML (ne contenant pas les balises HTML et BODY)
 - Un fragment de jsp
 - Du code java
- Permet de réutiliser le même fragment
- Avant génération du code de la servlet, le fragment est inclus

Une mini démo

- Prenons le fichier destinataires.htm suivant:

```
<ul>
```

```
<h3>
```

```
<li> Professeur</li>
```

```
<li> Etudiants SIG</li>
```

```
<li> Etudiants GI </li>
```

```
</h3>
```

```
</ul>
```

- Dans la JSP inclure ce fichier avec la directive:

```
- <%@ include file="destinataires.htm" %>
```

Les tags de scripting

- Ce type de tags permet d'insérer du code java dans la servlet générée à partir de la JSP.
- trois tags scripting
 - le tag de déclaration : Permet la déclaration de variables et de méthodes
 - le tag d'expression : évalue une expression et insère le résultat dans la page web générée.
 - le tag de scriptlets : permet d'en inclure le contenu dans la méthode service() de la servlet.

Les tags de déclaration

- Syntaxe :
 - `<%! Declaration %>`
- Permet de déclarer des variables et des méthodes
- Ne génère pas de code HTML dans la JSP

Les tags d'expression

- Syntaxe:
 - `<%= expression à évaluer%>`
- L'expression est évaluée et convertie en chaîne avec un appel à la méthode `toString()`
- Le résultat est inclus dans la page générée

Exemple

- Déclarons deux variables i et j et affichons leur somme

- insérer le code:

```
<%! int i1= 5; int i2=10;%>
```

```
<%! int somme(int i,int j)
```

```
{
```

```
return(i+j);
```

```
}%>
```

```
<%! int resultat=somme(i1,i2);%>
```

```
<h2> la somme de <%=i1%> et de <%=i2%> est: <%=resultat%>
```

```
</h2>
```


Les tags scriptlets

- Syntaxe:
 - `<% code java %>`
- Permettent d'insérer du code java dans la méthode service de la servlet générée
- Le code doit contenir exclusivement du java
 - Pas de balises HTML ou jsp

Exemple

- Rajouter le code suivant:

```
<%! int fact=1;%>
```

```
<% for(int i=1;i<=10;i++)
```

```
{%
```

```
<% fact*=i;%>
```

```
<H3> fact de <%=i%> est égal à <%=fact%>
```

```
</H3>
```

```
<%}%>
```

Les commentaires

- Pour les JSP, il est possible d'inclure deux types de commentaires
 - Les commentaires de la JSP
 - Syntaxe: `<%-- commentaire --%>`
 - Les commentaires du code HTML généré
 - Syntaxe: `<!-- commentaire -->`
- Introduire ces deux types de commentaires dans le code de votre JSP
- Conclure

Petit Exo

- Ecrire une JSP qui demande, sur la ligne de commande après affichage par `System.out.println`, deux entiers `a` et `b` et renvoie sur une page html le résultat de leur division.
- Introduire une JSP d'erreur pour prendre en considération la division par 0

Les EJBs



Un EJB, c'est quoi?

- Ecrit en Langage de programmation Java
- Correspond à la couche applicative
- Se situe du côté serveur
- Encapsule la logique business de l'application. Correspond au code qui réalise les objectifs (les besoins) fonctionnels.
- S'exécutent au sein d'un container EJB

Les atouts des EJBs: le container

- Différents avantages pour les développements des applications distribuées larges :
 - Le container est responsable des aspects niveau système
 - Gestion du cycle de vie
 - Communication,
 - Accès concurrents,
 - Transactions, la sécurité...
 - Le développeur se concentre sur la résolution des problèmes niveau business

Les atouts des EJBs: Portabilité et réutilisabilité

- Les composants EJBs sont des composants portables
- L'intégrateur de l'application peut la construire en intégrant et en composant des beans préexistants.
- Les applications peuvent s'exécuter sur n'importe quel serveur JEE du moment qu'elles utilisent les APIs standards

Les atouts des EJBs: Accès concurrents

- Les EJBs constituent des composants performants dans le cas où:
 - L'application possède plusieurs clients
 - Les clients peuvent localiser et interagir avec les beans
 - Les clients peuvent être légers, nombreux et de différents types

Les atouts des EJBs: Transactions

- L'application doit gérer des interactions transactionnels pour maintenir l'intégrité de données persistantes
 - Les EJB supportent les transactions
 - Leurs containers gèrent les accès concurrents des objets partagés et la consistance des bases de données adjacentes.

Les types des EJBs

- Session:
 - Réalisent des tâches pour le client
 - Peuvent implémenter des services Web
 - Traitent des interactions synchrones
- Message-Driven:
 - Réalisent des tâches pour le client
 - Traitent des interactions asynchrones
- Entity:
 - Représentent un objet relatif à une entité business avec un stockage persistant

Les beans Session



Tout d'abord, c'est quoi?

- Une instance d'un bean session représente un seul client dans le serveur d'application.
- Le client invoque les méthodes du bean session qui réalise les tâches business.
- Le fonctionnement de l'EJB session est similaire à celui d'une session interactive
 - Une instance est "dédiée à un seul client"
 - Elle n'est pas persistante
 - Quand le client se termine, l'EJB "se termine" et n'est plus associé au client
- Trois types de composants EJB session:
 - Singleton (singleton)
 - Sans état (stateless)
 - Avec état (stateful)

Les EJBs Session singleton

- L'EJB singleton est instancié une seule fois
- La durée de vie de son instance est la durée de vie de l'application
- La même instance est partagée en accès concurrent par tous les clients
- Maintient de l'état conversationnel entre les différentes invocations
- Un EJB singleton peut implémenter un service Web

Les EJBs session sans état (1/2)

- Ne maintient pas un état conversationnel pour le client
 - Pour une invocation de méthode, l'instance du bean peut contenir des variables avec état mais seulement pendant la durée de l'invocation.
 - A la fin de l'invocation, l'état est «perdu»
 - A part pendant l'exécution d'une invocation, toutes les instances sont "équivalentes"
 - Le container peut allouer n'importe quelle instance à n'importe quel client

Les EJBs session sans état (2/2)

- Offrent une meilleure scalabilité pour les applications comprenant un grand nombre de clients
 - Une applicationinstanciera moins de composants EJB sans état que de composants avec état pour servir le même nombre de clients
- Un composant EJB session sans état peut implémenter un service Web

Les EJBs session avec état

- L'état d'une instance d'un objet correspond aux valeurs de ses variables
- Ces variables représentent l'état d'une session unique client-bean.
- Cet état est appelé l'état conversationnel.
- L'état est maintenu pour la durée de la session.
- Si le client termine ou supprime l'instance du bean, cet état disparaît.

Cas d'utilisation d'un EJB session

- A n'importe quel moment un seul client accède à une instance de l'EJB
- L'état de l'EJB n'est pas persistant, il existe seulement pour une courte période
- Le composant EJB implémente un service Web

Cas d'utilisation d'un EJB session singleton

- Les EJBs Singleton sont utiles au cas où
 - L'état conversationnel doit être partagé par tous les clients
 - Le bean session doit réaliser des tâches d'initialisation une fois pour toute l'application
 - Le bean doit réaliser des tâches de nettoyage avant l'arrêt de l'application

Cas d'utilisation d'un EJB session sans Etat

- Pour augmenter la performance, le choix d'un EJB sans état est recommandé dans les cas suivants:
 - Pour une invocation de méthode, le composant réalise une tâche générique pour tous les clients
 - L'état du composant ne possède pas de données pour chaque client

Cas d'utilisation d'un EJB session avec Etat

- Un EJB Session avec état est recommandé dans les cas suivants:
 - L'état de l'EJB représente l'interaction entre le bean et un client spécifique
 - Le composant a besoin d'enregistrer de l'information entre plusieurs invocations du client
 - Le composant joue le rôle d'intermédiaire entre le client et les autres composants de l'application pour lui présenter une vue simplifiée

Les bean Entité



C'est quoi?

- Un bean entité représente un objet métier dans un mécanisme de stockage persistant.
 - Un consommateur
 - Un produit
 - Un compte bancaire
- Dans le serveur d'application, le mécanisme de stockage correspond à une base de donnée relationnelle
 - Chaque composant correspond à une table dans la base de données
 - Chaque instance correspond à une ligne de cette table

Différences avec les EJBs session

- Les EJBs entité sont persistants
- Permettent des accès concurrents
- Possèdent des clé primaires
- Peuvent participer à des relations avec d'autres EJBs entité

La persistance

- La persistance d'un EJB entité découle du fait qu'il est sauvegardé dans un mécanisme de stockage
- Cette persistance implique que l'état de l'EJB existe au-delà du processus d'exécution de l'application ou du serveur d'application
- L'état persiste même après l'arrêt du serveur de base de données ou de la machine sur laquelle il est déployé

Les types de persistance

- Pour les EJBs entité, il existe deux types de gestion de la persistance:
 - Bean-managed
 - Le code de l'EJB contient les appels d'accès à la base donnée (écrit par le développeur)
 - Container-managed
 - Le container de l'EJB génère automatiquement les appels d'accès à la base de données.
 - Le code de l'EJB ne contient pas ces appels

Les accès partagés

- Les EJBs entité peuvent être partagés par plusieurs clients
 - Les clients peuvent vouloir modifier les mêmes données
 - Il est important que les interactions avec les EJBs entité intègrent des transactions
 - Le container supporte la gestion des transactions
 - Les attributs des transactions sont spécifiés dans descripteur de déploiement du bean
 - Il n'est pas nécessaire de coder les frontières des transactions, c'est le container qui s'en charge

Les clés primaires

- Chaque EJB entité possède un identifiant unique d'objet
 - Numéro de compte
 - Numéro de pièce d'identité
 - Adresse mail...
- La clé primaire permet à un client de localiser les EJBs entité

Les champs persistants

- Les champs persistants d'un bean entité sont enregistrés dans la BD
- Collectivement, ils constituent l'état du bean
- Pendant la phase de déploiement, le container réalise
 - Un mapping du bean sur une table de base de données et,
 - Un mapping des champs persistants sur les colonnes de cette table
- Pendant l'exécution, le container du bean synchronise automatiquement cet état avec la BD

Cas d'utilisation des EJB entité

- Le type EJB entité est recommandé dans les cas suivants:
 - Le bean représente une entité business et non une procédure
 - Le bean doit garder un état persistant. Si son exécution se termine ou si le serveur d'application est arrêté, son état doit être maintenu dans une base de données



Les beans Message-Driven

C'est quoi?

- Un message-driven bean est un composant qui permet aux applications JEE de traiter les interactions asynchrones
- Agit comme un écouteur (listener) JMS
- Les messages peuvent être envoyés par n'importe quel composant JEE
 - Client applicatif
 - Un autre bean
 - Un composant Web ...

Caractéristiques

- Les instances d'un bean message-driven ne conservent ni données ni état conversationnel
- Toutes les instances d'un tel bean sont équivalentes. Le container EJB peut assigner n'importe quelle instance à un émetteur donné
- Un seul bean peut traiter les messages de différents émetteurs de manière concurrente

Fonctionnement

- Les messages-driven s'exécutent à la réception du message d'un client
 - Ils sont invoqués de manière asynchrone
 - À l'arrivée d'un message, le container appelle la méthode `OnMessage` du bean pour réaliser le traitement
 - Ont un cycle de vie court
 - Ils ne représentent pas de données partagées dans les BD mais peuvent y accéder
 - Ils peuvent considérer des transactions
 - Ils sont stateless

Cas d'utilisation des Message-driven

- Les interactions impliquant directement des beans session et entités, sont fondamentalement synchrones
- Afin d'optimiser les ressources du serveur, les beans message-driven sont un choix judicieux pour implémenter des interactions de manière asynchrones quand elles s'y prêtent.

Les EJB version 3



De la version 2 à la version 3

- La différence entre la version 2 et 3 des EJBs concerne principalement le modèle de programmation
 - Interfaces
 - Informations destinées au container
 - Accès aux beans
- Les fondements de la technologie restent les mêmes
 - Un framework de développement pour les composants côté serveur
 - Les concepts liés aux types de composants
 - Les principes architecturaux
 - Le protocole de communication...

Les ejb 3

- Composants métier sur le tier application
- Déployés dans le cadre d'applications distribuées multi-tier
- Un bean est implanté en utilisant l'API EJB (package javax.ejb.*)
- Il est déployé dans un container d'EJBs qui gère:
 - Cycle de vie
 - Transactions
 - Accès concurrents...
- Le client du bean peut être une Servlet, une JSP, un autre bean...
- Une tâche complexe est réalisée par un ensemble de beans
 - Le client invoque une méthode d'entrée sur un des beans
 - Ce bean invoque les autres beans qui coopèrent pour accomplir cette tâche

Le container

- Le container est responsable de
 - la gestion des beans
 - offrir un environnement d'exécution distribué, sûr, et transactionnel
- Ni le client, ni le bean ne sont astreints à prendre en compte du code pour interagir avec le container
 - Il doivent spécifier leurs besoins dans
 - le descripteur de déploiement à travers du code XML
 - ou dans le code du bean à travers des annotations

Les services du container (1/2)

- Le container agit comme un middleware entre le client et le bean.
- Il offre au bean des services implicites tels que
 - Gestion des ressources et du cycle de vie
 - Création des instances, destruction, passivation, activation
 - Threads
 - Sockets
 - Connections à la base de données

Les services du container (2/2)

- Accessibilité distante
 - Permettre aux clients l'accès à des méthodes sur des beans avec une JVM distante
- Support des requêtes concurrentes:
 - Permettre de servir des requêtes concurrentes sans que le développeur considère le multithreading dans le code
- La gestion des transactions
 - Exposé à travers l'API Java Transaction API (JTA)
- La sécurité,
 - Authentification et gestion des accès au code Java
 - EJB introduit aussi la notion de sécurité transparente
 - Mettre en place des attributs de sécurité au lieu d'utiliser l'API security

Les types des beans

- Les types pour la version 3
 - Les beans session
 - Stateless
 - Singleton
 - Stateful
 - Les beans entity
 - Les beans MDB

RMI-IIOP

- Internet Inter-ORB Protocol (IIOP) a été introduit par l'OMG pour permettre la communication entre ORBs à travers le net pour la technologie CORBA
- Les EJBs qui ont adopté le même protocole
- Tout container conforme à la spécification doit supporter RMI-IIOP
 - Mariage entre le protocole IIOP et le modèle de programmation et l'API RMI

Transparence de la localisation

- Objectif: permettre à des clients de communiquer avec des objets distants sans avoir à connaître la localisation de leur machine
- Généralement réalisée à travers un intermédiaire ou un registre
- Dans EJB, communiquer avec un objet se trouvant sur une autre JVM que celle du client
- JNDI est l'outil utilisé dans le monde des EJBs
 - Le client peut communiquer avec l'objet du moment qu'il peut communiquer avec le service JNDI du container
 - A partir de la version 3.1, l'injection de référence

Les annotations

- Les annotations, appelées aussi Métadonnées, permettent d'introduire des notions sémantiques sur le code
 - Au départ utilisées pour commenter du code pour un relecteur
 - Utilisées maintenant aussi pour être compilées ou interprétées par des compilateurs, des outils de déploiement et de développement (à partir de java 1.5)
- Les annotations peuvent être relatives à des méthodes, variables, constructeurs...
- Une annotation commence par le signe « @ » suivi par le nom de l'annotation et par les données s'il y a lieu
- Pour les EJB, cela permet de définir de manière plus simple des informations à l'intention du container



Exemple de code en
EJB3

Session Bean Stateless

L'interface business

- Une interface tout comme les autres, dans un fichier Bonjour.java:
 - package DireBonjour;
 - public interface Bonjour
 - {
 - public String bonjour();
 - }

La classe du bean

- Aussi une classe comme les autres qui implémente l'interface business, dans un fichier BonjourBean.java:
 - package DireBonjour;
 - import javax.ejb.Local;
 - import javax.ejb.Stateless;
 - @Stateless
 - @Local(Bonjour.class)
 - public class BonjourBean implements Bonjour
 - {
 - public String bonjour(){ return("Bonjour Les gars!");
 - }
 - }

Le client

- Dans un fichier LeClient.java:
 - package DireBonjour;
 - import javax.naming.Context;
 - import javax.naming.InitialContext;

 - public class LeClient{
 - public Context ctx=new InitialContext();
 - static void main() throws Exception{
 - Bonjour b=(Bonjour) ctx.lookup("DireBonjour.Bonjour");
 - System.out.println(b.bonjour());
 - }}

Session Bean Singleton



Interface du bean

- package DireBonjour;
- public interface Bonjour
- {
- public String bonjourAvecNom();
- }

La classe du bean

- Aussi une classe comme les autres qui implémente l'interface business, dans un fichier BonjourBean.java:
 - package DireBonjour;
 - import javax.ejb.Local;
 - import javax.ejb.Stateless;
 - @Singleton
 - @Local(Bonjour.class)
 - public class BonjourBean implements Bonjour
 - {
 - public String Nom="NomInitial";

La classe du bean

- Aussi une classe comme les autres qui implémente l'interface business, dans un fichier BonjourBean.java:
 - public void SetNom(String N)
 - {
 - Nom=N;
 - }
 - public String bonjour()
 - {
 - return("bonjour "+Nom);
 - }
 - }

Le client

- Ecrire un client sous forme de servlet utilisant une instance du bean
 - A chaque appel GET
 - Affichage d'un champ pour saisir une chaîne de caractères et un bouton sur le POST
 - A chaque appel POST
 - On affiche un message comprenant la valeur de l'attribut représentant l'état du bean
 - On positionne la valeur de l'attribut à la chaîne de caractères saisie
- Ecrire un deuxième client sous forme de servlet utilisant deux instances du bean
 - A chaque GET
 - Deux champs pour saisir deux chaînes de caractères
 - A chaque POST
 - Affichage des valeurs des attributs des deux instances
 - Positionnement des valeurs des attributs aux deux chaînes saisies

Les Stateful session beans



Gestion des instances par le container

- A la différence des session stateless, le container à moins de latitude pour la gestion des instances du stateful
 - À la première invocation d'un client, le container crée une instance dédiée
 - Tout au long de la conversation, le container doit garder l'état de l'instance pour chaque client
- Un problème de scalabilité se pose
 - Pour les stateless, ce problème ne se posait pas (toutes les instances sont équivalentes)
 - Ressources limitées Vs besoin de sauvegarder l'état de toutes les instances jusqu'à la fin des conversations

Gestion des instances par le container

- Dans les systèmes d'exploitation, ce problème est géré en faisant appel au disque comme extension de la mémoire physique (mémoire virtuelle)
- Les containers utilisent la même approche pour préserver les ressources
 - Pour limiter le nombre des instances en mémoire,
 - le container peut stocker leur état sur le disque (passivation)
 - Quand le client renvoie une nouvelle invocation, l'instance est recrée (activation) avec le même état conversationnel stocké

Gestion des instances par le container

- La décision de passiver une instance est spécifique à chaque container
 - La plupart utilisent une approche LRU (Least Recently Used)
 - Le bean qui a été invoqué le moins récemment est passivé
- Un bean peut être passivé à n'importe quel moment tant qu'il n'est pas en appel de méthode
- Une seule exception
 - Si le bean est impliqué dans une transaction, il ne peut être passivé tant que la transaction n'est pas terminée
- Pour l'activation, les containers utilisent généralement une approche just-in-time
 - Le bean est activé quand un client effectue une nouvelle invocation

Les méthodes de callback

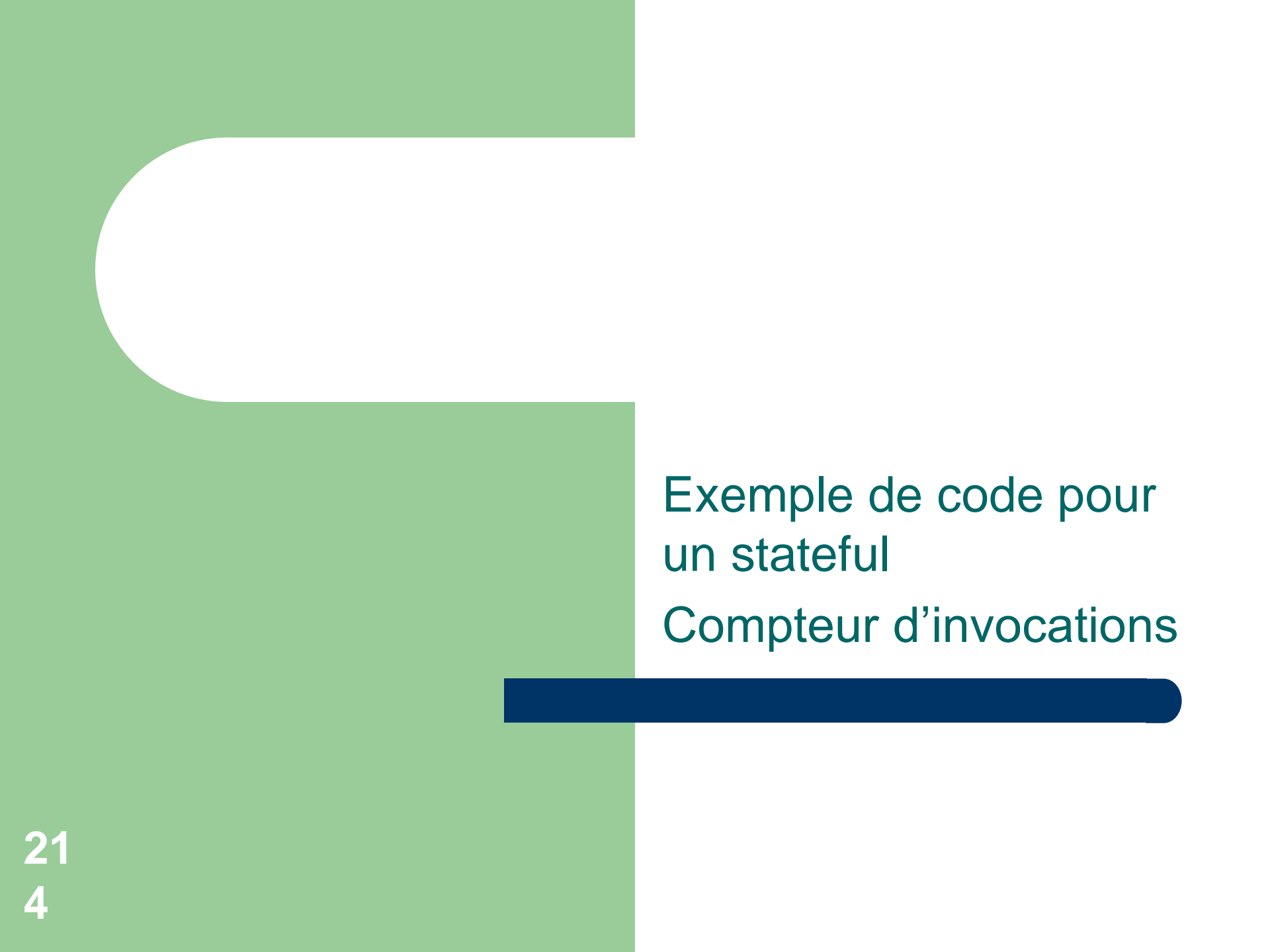
- Rappelez-vous des méthodes dans la classe du bean en version 2.1 (ejbCreate, ejbPassivate...)
- Ces méthodes possèdent leur utilité pour interagir avec le container
- Elle suivent les règles suivantes
 - Possèdent la forme public void method() dans la classe du bean
 - Possèdent la forme public void method(beanClass bean) dans un écouteur callback
 - Pas d'injection de dépendances dans les classes des écouteurs callback

Les 4 callback (1/2)

- Post création, appelée juste après la création d'une nouvelle instance
 - Annotation: `@PostConstruct`
 - Pour les stateless: permet d'initialiser le bean
 - Pour les stateful: idem
- Pre Passivation, appelée immédiatement avant de passer une instance
 - Annotation: `@PrePassivate`
 - Pour les stateless: inutile
 - Pour les stateful: permet de libérer des ressources utilisées par le bean

Les 4 callback (2/2)

- Post activation, appelée juste après l'activation d'une instance
 - Annotation: `@PostActivate`
 - Pour les stateless: inutile
 - Pour les stateful: permet de récupérer les ressources relâchées pendant la passivation
- Pre suppression, appelée immédiatement avant de supprimer une instance
 - Annotation: `@PreDestroy`
 - Pour les stateless: libérer les ressources allouée par le bean
 - Pour les stateful: idem



Exemple de code pour
un stateful
Compteur d'invocations

L'interface business

- package lesStatefuls
- public interface Compter
- {
- public int LeCompte();
- public void set(int val);
- public void remove();
- }

Remarques

- L'interface définit les méthodes qui vont être implantées dans la classe du bean
- L'interface est une interface POJO et n'hérite pas d'une autre interface du framework EJB
- L'aspect remote ou local n'est pas spécifié
 - n'hérite pas de `java.rmi.Remote`
 - Ne lève pas l'exception `java.rmi.RemoteException`

La classe du bean (1/2)

- package lesStatefuls;
- import javax.ejb.*;
- import javax.interceptor.*;
- @Stateful
- @Remote(Compter.class)
- public class CompterBean implements Compter
- {
- private int val;
- public int LeCompte()
- {
- System.out.println("Je compte!");
- return ++val;}

La classe du bean (2/2)

- `public void set (int v){`
- `this.val=v;`
- `System.out.println("Le compteur est à" + val);}`
- `@Remove`
- `public void remove(){`
- `System.out.println("l'instance est supprimée");`
- `}`

Remarques

- La classe du bean est une classe ordinaire avec des annotations
- Le fait d'implémenter les interfaces business est optionnel
 - Peut utiliser les annotations @Remote et @Local pour spécifier les interfaces concernées
 - Sans annotations, une interface est considérée comme locale
 - Si plusieurs interfaces, la spécification de leur type est obligatoire
 - L'annotation @Remove indique au container que le client a terminé sa conversation

Le client

- Les mêmes étapes
 - Acquérir un contexte initial JNDI
 - Récupérer une référence sur l'interface business
 - Nous ferons trois lookup pour avoir trois conversations avec trois instances
 - Appeler la méthode Count
 - Supprimer les instances

Le Client

- `package LesStatefuls;`
- `import javax.naming.*;`
- `public class LeClient{`
- `public static final int nbrClients=3;`
- `public static void main(){`
- `try{`
- `Context ctx=new InitialContext();`

Le Client

- `Compter Compteurs[]=new Compter[nbrClients];`
- `for (int i=0;i<nbrClients;i++)`
- `{`
- `Compteurs[i]=(Compter)`
`ctx.lookup("lesStatefuls.Compter");`
- `Compteurs[i].set(i);`
- `System.out.println(Compteurs[i].LeCompte());`
- `System.out.println(Compteurs[i].LeCompte());`
- `}`

Le Client

- for (int i=0;i<nbrClients;i++)
- {
- System.out.println(Compteurs[i].LeCompte());
- System.out.println(Compteurs[i].LeCompte());
- Compteurs[i].remove();
- }}
- catch(Exception e){
- e.printStackTrace();
- }}}

La persistance



Java Persistence?

- La spécification des composants entity bean est restée la même de la version 2.1 à la version 3.0 du framework
- La nouveauté est la présence d'une couche de persistance qui peut être utilisée par les entités
- C'est quoi?
 - Un document de spécification de 220 pages (séparé de la spécification EJB)
 - Permet une programmation POJO pour les objets persistants

Caractéristiques

- Offre un mapping modèle objet-modèle relationnel
- N'est pas lié à JEE et peut être utilisé dans un environnement JSE
- Définit une interface pour les fournisseurs de service (service provider) de la persistance permettant des invocations standards

Le mapping objet-relationnel

- Le moyen le plus simple d'implanter la persistance en Java est l'utilisation de la sérialisation native
 - Limitations
 - Recherche non efficace
 - Problèmes d'accès concurrents/transactions
- L'autre moyen est l'utilisation des bases de données avec des SGBD (Oracle, SQL Server, MySQL..)
 - Possibilité d'interroger ces systèmes via JDBC et créer un mapping manuellement (très fastidieux et couteux en développement)
 - Utiliser des produits pour réaliser ce mapping tels que TopLink Oracle ou Hibernate
 - Un mappeur automatique réalise le mapping

Les entités

- Les objets persistants sont appelés entités dans la spécification « Java Persistence »
 - Des objets POJOs (Plain Old Java Objects)
 - Stockent les données dans des champs
 - numeroDeCompte
 - Solde
 - Possèdent des méthodes associés à ces champs
 - getNumeroDeCompte()
 - getSolde()

Les entités Vs les bean session

- Les grandes différences
 - Les entités possèdent une identité persistante (la clé primaire) distincte de la référence de l'objet
 - Les entités possèdent un état persistant
 - Les entités ne sont pas accessibles de manière distante
 - Le cycle de vie d'une entité peut être indépendant du cycle de vie de l'application

Le provider de persistance

- Responsable du mapping objet-relationnel
- Constitue le mécanisme de transfert de et vers les objets java et la base de donnée
- En JEE, c'est le container qui joue le rôle du provider de persistance
 - S'occupe de déterminer le moment de charger et d'enregistrer les données
 - S'occupe de déterminer les besoins de rafraîchissement des instances
- Vous n'avez pas besoin de vous occuper de la synchronisation des objets, c'est le provider de persistance qui s'en occupe

La classe entité

- Les entités sont similaires aux composants EJB
 - Une classe Java
 - Des métadonnées sous forme d'annotations
 - Un descripteur de déploiement XML
- Elles peuvent être utilisées dans un contexte JEE ou JSE

Exemple: classe de l'entité

- package entites;
- import java.io.Serializable;
- import javax.persistence.Entity;
- import javax.persistence.Id;
- @Entity
- public class Compte implements Serializable{
- @Id
- private int NumeroCompte;
- private int Solde;

Exemple

- `public Compte(){`
- `NumeroCompte=(int) System.nanoTime();`
- `}`
- `public void crediter(int val){`
- `Solde+=val;`
- `}`
- `}`

Remarques

- La classe de l'entité est une classe java qui n'hérite d'aucune classe
 - Ici, c'est juste pour permettre le stockage simple et surtout pour que sa référence puisse être envoyée comme argument des invocations distantes
- Correspond
 - à une définition de données dans une base relationnelle (une instance correspond à une ligne)
- Une entité doit déclarer une clé primaire avec l'annotation @Id
- L'accès à l'état de l'entité se fait directement par l'accès aux attributs mais il est possible de définir des getters et des setters
- L'entité peut exposer des méthodes business pour manipuler ou accéder aux données

Accès aux entités via un contexte de persistance

- Une entité n'est pas accessible de manière distante
- La solution est de la déployer localement et de l'invoquer à travers un bean session ou un MDB déployé sur le même container
- Pour cela, il faut intégrer des notions suivantes
 - L'objet « persistence context »
 - permettant la connexion entre les instances en mémoire et la base de données
 - L'interface « entity manager »
 - L'API permettant de manipuler le « persistence context »

Exemple: un session stateless pour accéder à une entité

- package entites;
- import java.util.List;
- import javax.ejb.Stateless;
- import javax.persistence.PersistenceContext;
- import javax.persistence.EntityManager;
- import javax.persistence.Query;
- @Stateless
- @Remote(Banque)
- public class BanqueBean implements Banque{

Exemple

- @PersistenceContext
- private EntityManager manager;
- public List<Compte> listAccounts(){
- Query q=manager.createQuery("SELECT a FROM Compte a");
- return q.getResultList();
- }
- public int getSolde(int num)
- {
- Compte c=manager.find(Compte.class,num);
- return c.Solde;
- }

Exemple

- `public Compte ouvrir(int s)`
- `{`
- `Compte c=new Compte();`
- `c.Solde=s;`
- `Manager.persist(c);`
- `return c;`
- `}`
- `public void fermer(int num)`
- `{`
- `Compte c=manager.find(Compte.class,num);`
- `manager.remove(c);`
- `}`
- `public void crediter(int num, int somme)`
- `{`
- `Compte c=manager.find(Compte.class,num);`
- `c.crediter(somme);`
- `}`
- `}`

Remarques

- Deux niveaux de création
 - Créer un objet correspondant à une instance de l'entité (appel du constructeur)
 - Planifier sa synchronisation persistante avec la base (appel via l'interface EntityManager de la méthode persist): état managed
- Dans le cas du stateless (par défaut et sans annotations)
 - Le « persistence context » possède une durée de vie égale à la durée de la transaction
 - L'attribut transaction possède la valeur « required »: tout appel de méthode est exécuté dans le cadre d'une transaction (existante ou nouvelle)
 - Le « persistence context » termine avec la fin de l'invocation
 - Le lien entre entités et « entity manager » est supprimée: état detached

Types de contexte de persistance

- Un contexte de persistance associé avec un « entity manager » peut être de deux types
 - Transaction scoped
 - Utilisé avec des beans stateless
 - Durée de vie égale à la transaction
 - « detached » à la fin de la transaction
 - Extended
 - Utilisé avec des beans stateful
 - Durée de vie dure jusqu'à suppression par le container
 - Managed jusqu'à ce moment là

Exemple: accès via un stateful (1/3)

- package entites
- import javax.ejb.Remote;
- import javax.ejb.Stateful;
- import javax.persistence.EntityManager;
- import javax.persistence.PersistenceContext;
- import javax.persistence.PersistenceContextType;
- @Stateful
- @Remote(CompteRemote.class)
- Public class Compte implements CompteRemote{

Accès via un stateful (2/3)

- `@PersistenceContext(type=PersistenceContextType.EXTENDED, unitName="toto")`
- `private EntityManager manager;`
- `private Compte cmpt=null;`

- `public void ouvrir(int num)`
- `{`
- `cmpt=manager.find(Compte.class,num);`
- `if(cmpt==null){`
- `cmpt=new Compte();`
- `cmpt.numero=num;`
- `manager.persist(cmpt);}`

Accès via un stateful (3/3)

- `public void crediter(int somme)`
- `{`
- `if(cmpt==null)`
- `System.out.println("compte inexistant");`
- `else`
- `cmpt.deposer(somme);`
- `}`
- `}`

Remarques

- Nous utilisons l'annotation EXTENDED pour le contexte de persistance
- Nous gardons une instance de l'entité
- La combinaison de ces deux aspects permet
 - de garder l'instance dans l'état managed
 - de garder le contexte de persistance au-delà de la transaction (supprimé à la suppression du bean)
 - De ne pas appeler la méthode « find » avant chaque utilisation du bean (juste vérifier que c'est fait une fois durant la session)

Quel choix faire?

- Invocations multiples sur une même instance pendant une session
 - Bean session: stateful
 - Contexte de persistance: EXTENDED
- Invocation unique ou invocations multiples sur différentes instances
 - Bean session: stateless
 - Contexte de persistance: TRANSACTION

Les unités de persistance (1/2)

- Les entités sont déployées dans des unités de persistance (persistence unit)
- Unités de persistance
 - Un groupe logique d'entités
 - Des méta-données pour le mapping
 - Des données de configurations liées aux bases de données
- Définies avec un fichier de description
 - Persistence.xml

Les unités de persistance (2/2)

- Exemple simple
 - `<?xml version="1.0" encoding="UTF-8"?>`
 - `<persistence xmlns="http://java.sun.com/xml/ns/persistence">`
 - `<persistence-unit name="toto"/>`
 - `</persistence>`
- Il faut fournir au moins une unité de persistance
- Il est possible d'en définir plusieurs (c'est le nom qui va permettre d'indiquer lequel est utilisé)

L'API EntityManager

- C'est l'interface permettant d'accéder aux entités dans le contexte persistance de l'application
- Deux options pour le client pour utiliser l'EntityManager
 - Un « container managed » EntityManager
 - C'est le container qui se charge de déterminer et fournir un EntityManager à l'application
 - Injection via l'annotation `@PersistenceContext`
 - De la « SessionContext » via la méthode « lookup »
 - Un « application managed » EntityManager
 - C'est l'application qui est responsable de créer une instance via l'interface EntityManagerFactory

Les méthodes de l'interface

- L'API offre 3 types d'opérations:
 - Gestion de cycle de vie
 - Opérations de synchronisation avec la base de données
 - Recherche et requêtes sur les entités

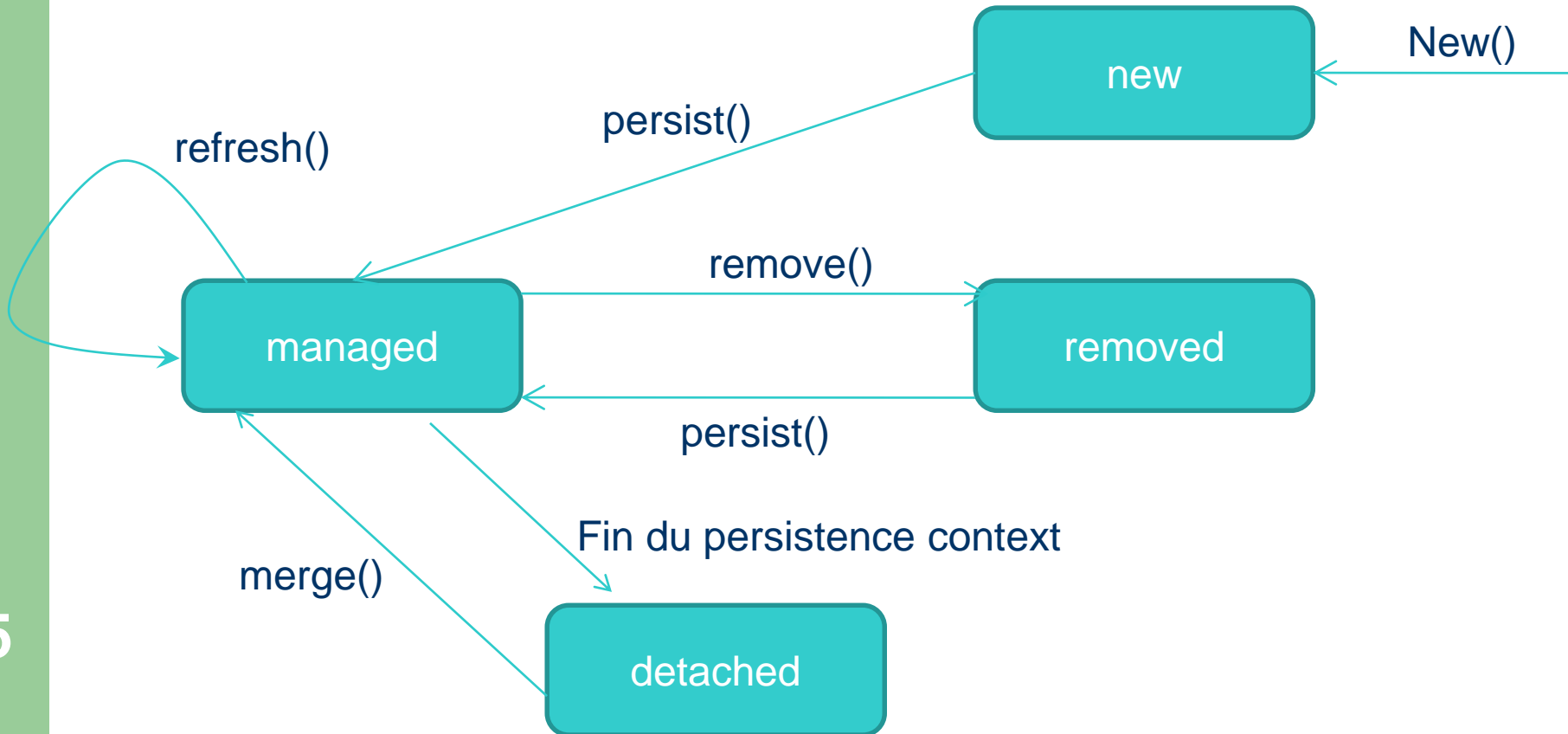
Cycle de vie d'une entité

- L'« EntityManager » distingue 4 états dans le cycle de vie d'une entité
 - New:
 - Etat correspondant à l'entité juste après l'appel du constructeur
 - l'instance est créée en mémoire
 - Elle n'est pas associée à une identité persistante ni dans un « persistence context » ni dans la base de données
 - Les changements dans l'état de l'entité ne sont pas synchronisés avec la base de données

Cycle de vie d'une entité (2/2)

- Managed
 - Correspond à l'état juste après l'appel de la méthode persist
 - L'entité est associée à un « persistence context » et possède une identité persistante dans la base de données
 - Les changements de l'état de l'entité sont synchronisés
 - à la fin des transactions
 - À l'appel de l'opération flush()
- Detached
 - L'entité possède une identité persistante mais n'est associée à aucun « persistence context »
- Removed
 - L'entité est associée à un « persistence context » mais est programmée pour être supprimée de la base de données

Etats et transitions d'une entité



Les callbacks

- Les entités peuvent aussi déclarer des méthodes callback
 - Dans la classe de l'entité
 - À travers un intercepteur
- Ces méthodes sont appelées par le provider de persistance
- Elles sont spécifiées en utilisant des annotations

Les événements traités par les callbacks

- L'API définit les événements (et les annotations correspondantes) suivantes
 - PrePersist
 - PostPersist
 - PreRemove
 - PostRemove
 - PreUpdate
 - PostUpdate
 - PostLoad
- Exemple:
 - @PrePersist
 - void AvantEnregistrement(){
 - System.out.println("l'entité va être synchronisée avec la base");}

Possibilité de définir une classe Listener

- Exemple
 - Dans la classe de l'entité
 - @Entity
 - @EntityListeners(Ecouteur.class)
 - public class Compte{....
 - Dans la class Ecouteur
 - @PrePersist
 - void AvantEnregistrement(Compte c) {
 - System.out.println("l'entité va être synchronisée avec la base");
 - }

Les méthodes de synchronisation

- Par défaut, les entités sont synchronisées avec la base de données quand la transaction termine
- Parfois, il est nécessaire de synchroniser à l'intérieur de la transaction
 - Permettre aux requêtes de prendre en compte le nouvel état
 - Pour réaliser cela, deux méthodes
 - flush(): permet de synchroniser toutes les entités du persistence context
 - refresh (Object entité): permet de rafraichir une entité à partir de la base de données

Les méthodes de recherche et de requête

- Généralement des données persistantes existent avant le lancement de l'application
 - Besoin de rechercher les entités déjà existantes
 - Besoin d'interroger la base de données avec des requêtes
- L'API de l'interface du manager offre la méthode de recherche
 - `<T> t find(Class<T> c, Object ClePrimaire)`
 - Permet de retrouver une entité par sa clé primaire

Invoquer des requêtes

- L'API EntityManager offre un moyen de réaliser des requêtes (EJB-QL ou SQL) via deux étapes
 - Création de la requête
 - `public Query createQuery(String requete_ejbql)`
 - `public Query createNativeQuery(String requete_sql)`
 - Exécution de la requête
 - `getResultList`
- Exemple:
 - `public List<Compte> lister(){`
 - `Query q = manager.createQuery("SELECT c FROM COMPTE c");`
 - `return q.getResultList();}`

Les EJBs et les transactions



Problématique

- L'exemple du virement bancaire illustre bien le besoin des opérations atomiques
 - Les deux opérations de débit et de crédit doivent être considérées comme une seule opération atomique
 - Risque de crash au niveau réseau
 - Risque de crash au niveau de la base de données
 - Les accès concurrents posent aussi problème au niveau de la consistance des données

Avantages des transactions

- Une transaction est une séquence d'opérations qui peuvent être vues comme une seule grande opération atomique
 - Garantie la proposition tout-ou-rien: toutes les sous opérations seront exécutées ou aucune ne le sera
 - Permet les accès concurrents: garanti que tous les utilisateurs partagent les mêmes données et que les actions d'écriture se font sans chevauchement

Les propriétés ACID

- 4 propriétés sont garanties par les transactions résumées dans l'acronyme ACID
 - Atomicity: les opérations sont toutes exécutées ou aucune ne l'est
 - Consistency: l'état du système est consistant après la fin de la transaction
 - Isolation: absence d'interférences entre les transactions en cours d'exécution
 - Durability: garantit la mise à jour des données même après les crashes (machine, réseau...) en utilisant des logs

Les Modèles transactionnels

- Plusieurs modèles transactionnels ont été définies.
- Deux modèles principaux
 - Les transactions plates (Flat transactions)
 - Les transactions encapsulées (nested transactions)

Modèle transactionnel plat

- Le modèle le plus simple
 - Une transaction plate = une série d'opérations exécutées d'une manière atomique
 - Après le début de la transaction
 - Exécution de plusieurs opérations (persistantes ou non)
 - A la fin de la transactions, un résultat binaire
 - Succès => commit: **Toutes** les opérations persistantes deviennent des changements permanents
 - Echec => rollback: **Toutes** les opérations persistantes sont défaites automatiquement

Modèle transactionnel encapsulé

- Le modèle encapsulé permet une plus grande souplesse dans le traitement des transactions
 - Une unité peut encapsuler d'autres unités
 - Le commit de toutes les unités implique le commit de l'unité mère
 - Le rollback d'une unité n'implique pas forcément le rollback de l'unité mère.
 - L'unité fille peut être réessayée et si elle réussit, l'unité mère réussit
 - Une transaction encapsulée peut être vue comme une arborescence de sous-transactions

Les transactions distribuées

- Les transactions peuvent concerner des composants distribués sur plusieurs tiers de déploiement impliquant plusieurs types de ressources
- Des points à prendre en compte dans ce cas, possibilité d'avoir
 - Plusieurs serveurs d'application coordonnant une transaction
 - Plusieurs bases de données
- Besoin de faire collaborer des entités logicielles à travers plusieurs processus, et réseaux
- Intervention du protocole two-phase commit (2PC) pour le support des transactions distribuées

Durabilité et distribution

- Rappel: permet de garantir que toutes les mises à jour qui sont en commit sont permanentes
- Avec plusieurs gestionnaires de ressources, besoin de mécanismes de réparation (si un gestionnaire crash)
- Un log est utilisé pour réaliser cet objectif et permettre une réparation si un crash intervient lors de la mise à jours

Le protocole 2PC

- Deux phases
 - Phase une:
 - commence par l'envoi d'un message « avant commit »
 - dernière chance pour les ressources de faire échouer la transaction
 - Si une des ressources échoue, toute la transaction est supprimée, pas de mise à jour des données
 - Sinon, la transaction continue et ne peut être stoppée
 - Toutes les mises à jours sont logées dans un journal persistant
 - Phase deux:
 - Réalisée seulement si la phase une est complétée. Tous les gestionnaires de ressources réalisent les mises à jour de données

Object Transaction Service

- La technologie EJB réutilise les concepts définies dans CORBA pour l'implantation des services de support des transactions
- Le service standard spécifié par l'OMG s'appelle OTS (Object Transaction Service)
 - Service optionnel de CORBA
 - Un ensemble d'interfaces utilisé par les transactions managers, les gestionnaires de ressources, les objets transactionnels pour collaborer
- Le service JTS utilisé dans les EJBs comme une couche au dessus de OTS

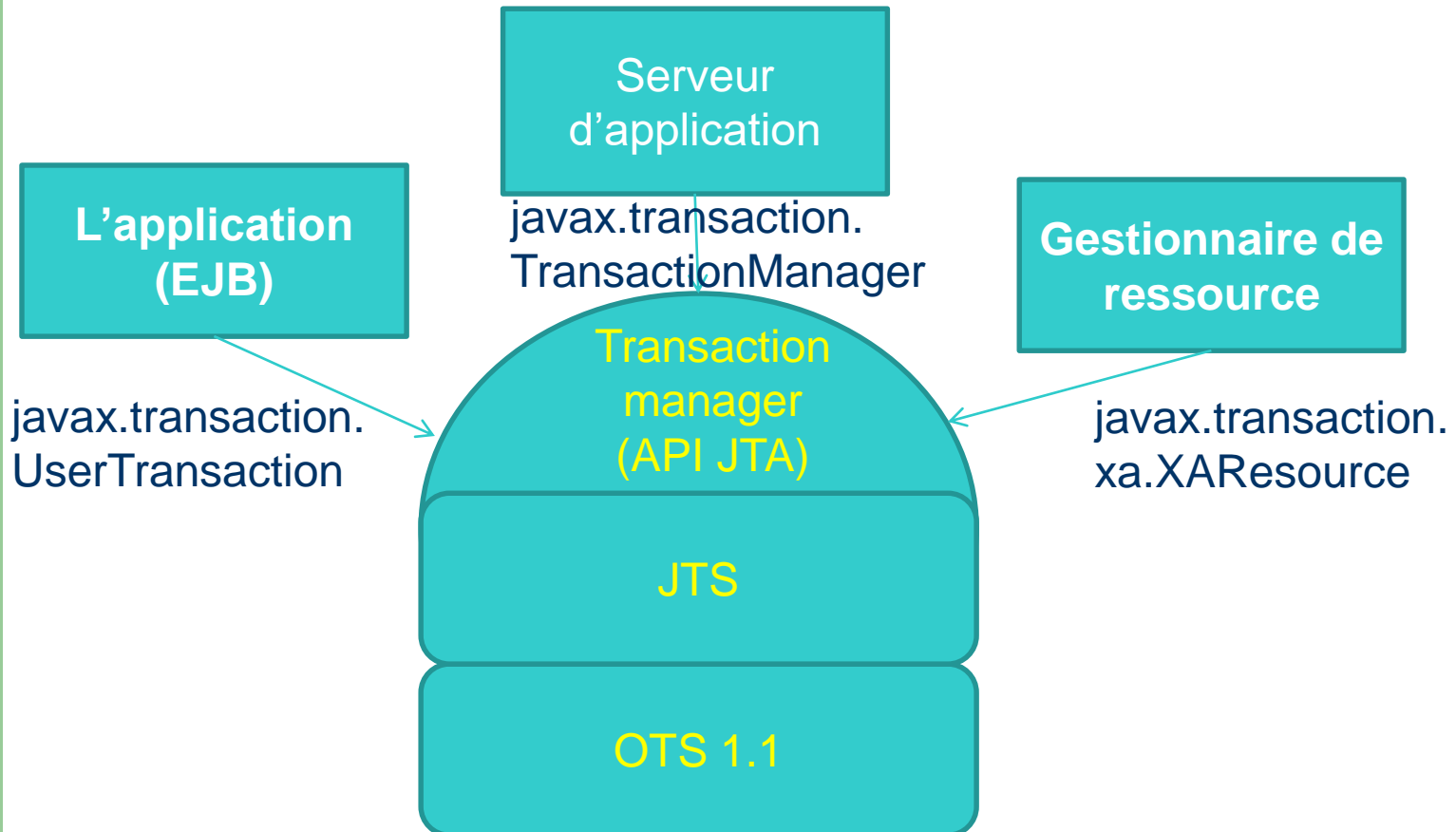
L'API Java Transaction (JTA)

- Constitue le support des transactions dans le monde des EJBs
- Définit les interfaces entre le transaction manager et les autres parties (beans, gestionnaires de ressources, serveurs d'application)
- Les communications entre le transaction manager qui implémente JTA et JTS passent à travers des interfaces propriétaire (mais JTA reste standard)

Les interfaces JTA

- JTA est constituée de trois interfaces
 - `javax.transaction.xa.XAResource`: interface utilisée par JTA pour communiquer avec les gestionnaires de ressources
 - `javax.transaction.TransactionManager`: interface utilisée pour communiquer avec le serveur d'application
 - `javax.transaction.UserTransaction`: interface utilisée par les beans et les programmes applicatifs pour intégrer les transactions EJB

Les couches transactions



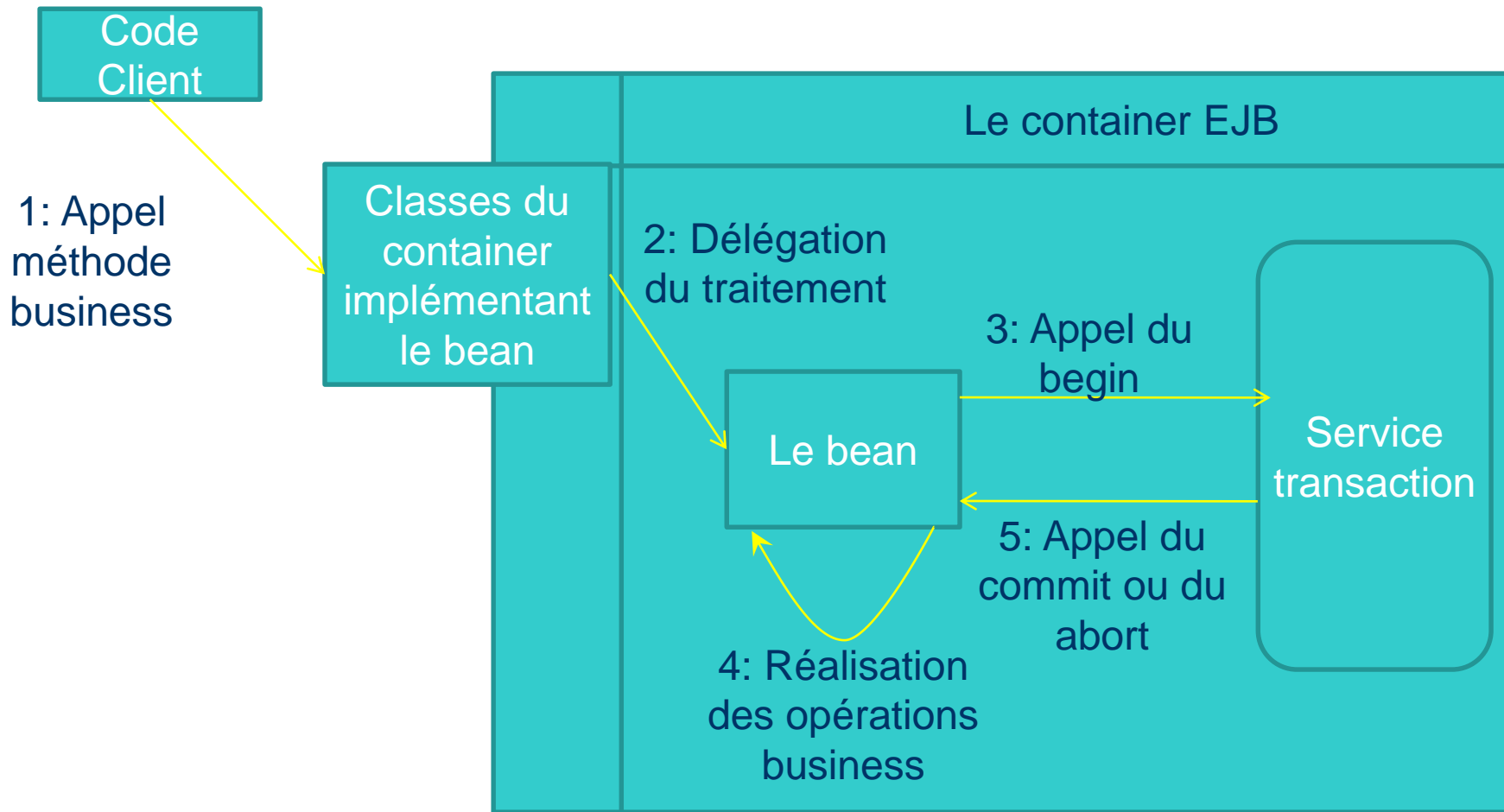
Le système d'abstraction des transactions

- La gestion des transactions aux niveaux les plus bas (OTS et JTS) est de la responsabilité du container
- Les beans n'interagissent pas avec le transaction manager ou les gestionnaires de ressources (standardisation et portabilité)
- Les beans peuvent juste spécifier si une transaction doit réussir ou échouer

Détermination des frontières pour les transactions

- une transaction: début puis
 - Ou bien: commit
 - Ou bien: rollback
- Points déterminants
 - Qui démarre la transaction?
 - Qui décide l'échec ou la réussite de la transaction
 - A quel moment interviennent ces étapes (frontières de la transaction)
- Pour les EJBs, trois acteurs de détermination
 - Les beans (bean-managed)
 - Le container (container-managed)
 - Le client (client-controlled)

Bean-managed

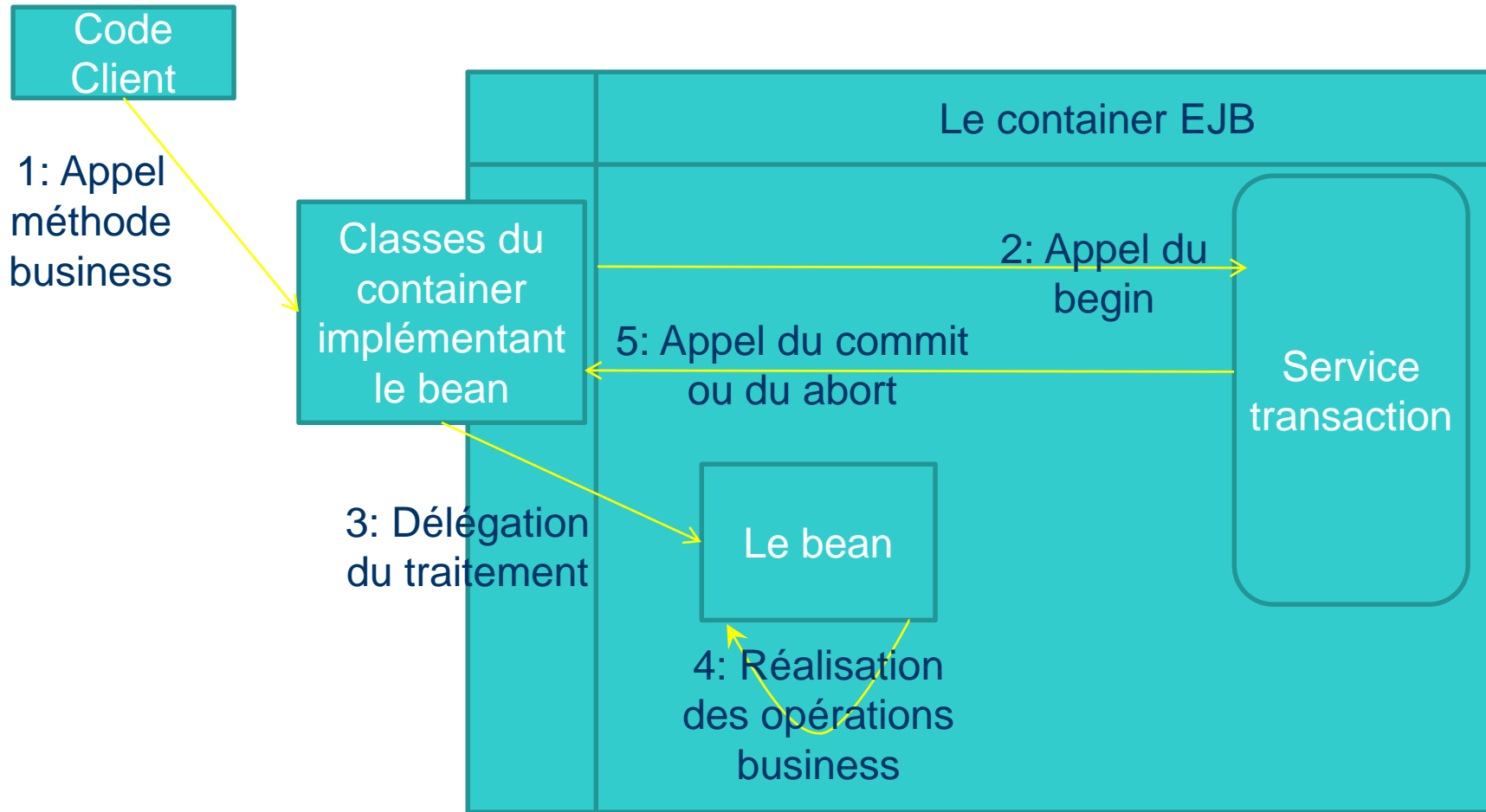


Container-Managed

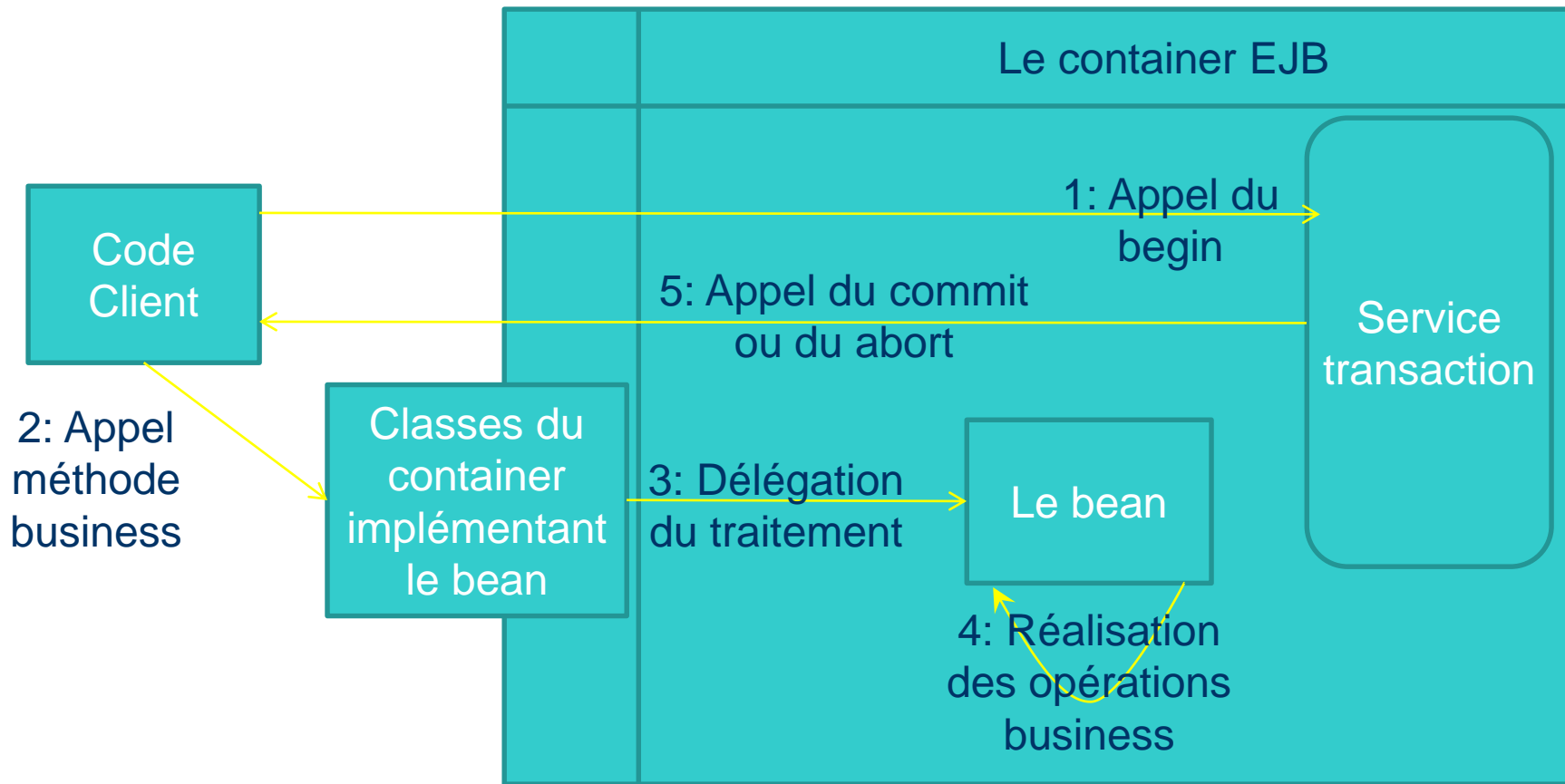
- Le mode par défaut
- Le bean ne réalise pas explicitement les appels « begin », « commit », ou « rollback »
- Le container définit les frontières des transactions en se basant sur le comportement transactionnel spécifié par le bean provider (annotations ou descripteur de déploiement)

-

Container-managed



Client-controlled



Le choix du style

- **Bean-managed**
 - Le développeur possède un contrôle total
 - Possibilité de définir plusieurs mini transactions à l'intérieur du même méthode
- **Container-managed**
 - Robustesse
 - Allègement et gain en temps de développement
- **Client-controlled**
 - Permet de prendre en compte les défaillances sur les communications client-bean (RemoteException)
 - Si crash impossibilité de savoir si la transaction a réussi ou a échoué sans rajouter du code

Le transactions gérées par le container

Attributs des transactions EJB

- Pour le cas Container-Managed Transactions, la gestion est spécifiée par le développeur via
 - des annotations: `@TransactionAttribute`
 - le descripteur de déploiement: balise `<trans-attribute>`
- Six attributs sont définis par la spécification EJB et doivent être implantés par les containers

Valeurs de attributs

- Required:
 - permet de spécifier que le bean s'exécute toujours à l'intérieur d'une transaction
 - Si une transaction existe déjà, il s'inscrit dedans
 - Sinon, une nouvelle transaction est démarrée par le container
- RequiresNew:
 - Exige la création d'une nouvelle transaction pour le bean
 - Si une transaction existe déjà, elle est suspendue
 - Après la fin de la transaction du bean, celle-ci peut reprendre

Valeurs des attributs

- Supports:
 - Si une transaction existe déjà, le bean s'exécute à l'intérieur de cette transaction
 - Sinon, il s'exécute en dehors de toute transaction
- Mandatory
 - Le bean doit s'exécuter dans une transaction pré-existante
 - Si une transaction existe déjà, le bean s'exécute à l'intérieur de cette transaction
 - Sinon, l'exception `javax.ejb.EJBTransactionRequiredException` est levée

Valeurs des attributs

- NotSupported
 - Le bean ne peut être impliqué dans aucune transaction
 - Si une transaction existe, elle est suspendue jusqu'à la fin de l'exécution du bean
- Never
 - Le bean ne peut être impliqué dans aucune transaction
 - Si une transaction existe, le container lève l'exception `javax.ejb.EJBException`

Résumé

Attribut	Transaction du client	Transaction du bean
Required	Aucune	T2
	T1	T1
RequiredNew	Aucune	T2
	T1	T2
Supports	Aucune	Aucune
	T1	T1
Mandatory	Aucune	Exception
	T1	T1
NotSupported	Aucune	Aucune
	T1	Aucune
Never	Aucune	Aucune
	T1	Exception

Exemple CMT

- `import javax.ejb.*;`
- `import javax.annotation.Resource;`
- `import javax.persistence.PersistenceContext;`
- `import javax.persistence.EntityManager;`

- `@Stateless`
- `@TransactionManagement(javax.ejb.TransactionManagementType.
CONTAINER)`
- `public class banqueBean implements banque{`
- `@PersistenceContext private EntityManager em;`
- `@Resource private SessionContext ctx;`

Exemple CMT

- @TransactionAttribute(javax.ejb.transactionAttributeType.Required)
- public void virement(float montant, int A_Debiter, int A_Crediter)
- {
- Compte c1=em.find(Compte.class,A_Debiter);
- Compte c2= c1=em.find(Compte.class,A_Crediter);
- if(c1.Solde<montant)
- {ctx.setRollbackOnly();}
- c1.retrait(montant);
- c2.depot(montant);
- em.persist(c1);
- em.persist(c2);
- }}

Descripteur de déploiement équivalent

- <?xml version="1.0" encoding=" UTF-8" ?>
- <ejb-jar
- xmlns=" http://java.sun.com/xml/ns/JEE"
- xmlns:xsi =" http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://java.sun.com/xml/ns/JEE
- http://java.sun.com/xml/ns/JEE/ejb-jar_3_0.xsd"
- Version="3.0" >
- <entreprise-beans>
 - <session>
 - <display-name>banqueBean</display-name>
 - <ejb-name>banqueBean</ejb-name>
 - <business-remote>banque</business-remote>
 - <ejb-class>banqueBean</ejb-class>
 - <session-type>Stateless</session-type>

Descripteur de déploiement équivalent

- `<transaction-type>Container</transaction-type>`
- `</session>`
- `<entreprise-beans>`
- `<assembly-descriptor>`
 - `<container-transaction>`
 - `<method>`
 - `<ejb-name>banqueBean</ejb-name>`
 - `<method-name>virement</method-name>`
 - `</method>`
 - `<trans-attributes>Required<trans-attribute>`
 - `</container-transaction>`
- `</assembly-descriptor>`
- `</ejb-jar>`

FIN

Merci pour votre attention

