Langages et Environnements Évolués
Lexique

8 octobre 2018

Lexique (probablement non exhaustif) des différents mots techniques vus en cours. Contient aussi d'autres mots importants de la sphère Java que l'on n'a pas pu voir en cours. N'hésitez pas à me demander si vous voulez que j'en rajoute.

1 Acronymes et mots techniques

Agile (hors programe) **:** Le développement agile est une méthode de management pour des projets en petite et moyenne équipe (4-20 personnes) et de durée courte et moyenne (1 mois à 1 an). Ce n'est pas un framework, une techno ou un design pattern, cela ne fait donc pas partie du cours.

AOP : *Aspect Oriented Programming.* [1] Il s'agit de retirer les aspects du corps des méthodes, les rendant parenthétiques. Pour ça, on utilise un framework dédié (comme Spring AOP) ainsi que des annotations qui vont lier la méthode sur le code de l'aspect en question.

API : *Application Programming Interface.* Il s'agit d'une définition de standards. En Java, une API (ou Java API) fait référence à une interface définie par Oracle. Généralement, ces interfaces ont une implémentation par défaut, ce qui fait qu'une API est utilisée comme une librairie (ou plus exactement une composante fonctionnelle autonome d'une librairie, qui elle contient plusieurs APIs qui peuvent interagir ensemble). Mais puisqu'il s'agit d'interface, elles ont généralement d'autres implémentations, que l'on peut trouver dans des librairies plus exotiques.

Bean : Un grain... [2] Seul, il désigne vaguement une petite composante de programmes qui ressemble à une classe mais qui va être passé à la moulinette via une pré-compilation. Avec un qualitatif, il va avoir un vrais sens technique (EJB, Java Bean, Spring Bean, NetBean...)

Builder (hors programe) **:** Les gros projets java utilisent un builder (comme Maven ou Gradle). Un projet qui utilise un builder contient un fichier de configuration spécifique qui décrit ce build (par exemple `pom.xml` pour Maven) ; il faut voire le fichier pom.xml comme un makefile haut niveau. Ce fichier de configuration permet non seulement de renseigner et de récupérer vos librairies, mais il va aussi référencer vos fichiers de codes et de configuration. A noter que les IDEs avancés (intelliJ, Eclipse, Netbin) contiennent des builders intégré dont il remplissent automatiquement la configuration, ce qui fait que l'on n'est pas obligé d'en utiliser un explicitement pour des petit projets (mais cela devint nécessaire lorsque l'on travail avec des personnes n'utilisant pas le même IDE ou lorsque l'on veut tweecker le build).

CRUD (hors programe) **:**

Design pattern (hors programe) **:** Le mot design pattern regroupe des standards de codes de types très différents, comme les différents schémas d'architecture, les layers classiques d'applications (entités, DAO, DTO...), et des solutions standards aux limites de java (Factories...).

Eager : C'est l'inverse de lazy. On charge tout par défaut.

Framework : Un framework est une extension du langage (pour nous de Java) qui peu appliquer des changements majeurs à la compilation du programme. En particulier de nombreuses taches administratives sont implicitement déléguées au framework. Celui-ci est donc souvent dédié à une (ou quelques) tâche(s) spécifique(s).

Frontend/Backend(/middlend) : Lorsque l'on a une architecture 2-tier (cf. N-tier), le premier tier est généralement appelé frontend et le second backend. Sur une 3-tier, on peut parler de frontend-middlend-backend. Par habitude, lorsque l'on a une architecture n-tier (pour $n \geq 3$, on parle souvent de frontend pour le premier tier et de backend pour tous les autres, l'idée étant que le frontend correspond à l'interfaçage avec le client.

1. La comparaison implicite avec le paradigme objet est un peu fort ici...
2. Référence au grain de café, que l'on peut moudre pour faire du java (cf. logo java). La communauté java des années 90 ont épuisé la métaphore...

IDE : *Integrated Developement Environement.* Ou environnement de développement. Il s'agit d'un logiciel de traitement de texte extrêmement évolué permettant de faciliter le code dans les langages de programmation supportés. InelliJ, PyCharm, Netbean, Eclipse, Atom sont des IDE très utilisés.

Interface (en java) : Il s'agit d'un concept de base de Java : une interface définit un type, les objets d'une classe implémentant l'interface sont de ce type, pour implémenter l'interface une classe doit implémenter certaines méthodes. Une interface est souvent vue comme un contrat dont les termes sont les méthodes à implémenter.

IoC : *Inversion of Control.* L'inversion de contrôle est un concept très utilisé par les frameworks. Il s'agit de permettre au contexte d'exécution d'accéder à un programme et d'en modifier la compilation. L'IoC est particulièrement explicite dans Spring qui permet de définir vos propres hooks sur les programmes.

IoD : Injection of Dependancies. L'injection de dépendance est une des formes les plus simples d'inversion de contrôle. Il s'agit de générer automatiquement des dépendances de programmes en choisissant quelles dépendances utiliser en fonction du contexte d'utilisation.

Java Beans (hors programe) **:** A ne pas confondre avec EJB. Un Java Bean est un composant java classique interagissant avec un servlet dans un cadre non JEE.

Journalisation : *Logging.* C'est le terme associé à la production de logs.

JVM (hors programe) **:** *Java Virtual Machine.* Il s'agit de l'interpreteur de bitcode Java.

Lazy : La programmation paresseu est un "style" de programmation où l'on ne charge *que* ce dont on a besoin *quand* on en a besoin. Cela veut dire que l'on ne charge pas trop, par contre, cela veut aussi dire que l'on fera des requêtes plus souvent. Par défaut, JPA et Hibernate sont en *lazy loading* ce qui veut dire qu'ils font leurs requêtes SQL en mode lazy.

Log : Un log est l'enregistrement d'informations concernant l'état d'exécution du programme. Un log peut être une `Trace` (simple information sur un point de passage), une `info` (l'état est inhabituel), un `warning` (l'état du programme n'est pas celui prévu), une `error` (une méthode n'a pas pu aboutir) ou une erreur `fatal` (l'appli a planté). Ces logs sont répartis sur plusieurs canaux (mail, serveur, bdd...) selon leur gravité et les options de compilation.

Oracle (hors programe) **:** Entreprise gérant le langage Java.

ORM : *Object Relational Mapping.* Il s'agit d'un type de framework qui crée une base de données virtuelle en paradigme objet et la lie avec une base de données réel en paradigme relationnel. L'application java n'a alors qu'a interagir avec la base de données objet à l'aide d'un langage dédié (comme JPQL ou HQL). Exemples d'ORM : JPA, Hibernate.

Paresseux : cf. Lazy

Schéma d'architecture (de programme) : Il s'agit d'un *design pattern* décrivant la structure globale d'un programme (ou d'un bout de programmes). En Java, il s'agit de l'organisation des classes, ou de groupes de classes. Attention, en changeant de granularité, un programme peut contenir plusieurs schémas d'architecture.

Serialization : Interuption et enregistrement d'un état courant dans un fichier/BdD. En Java, c'est la sauvegarde d'un objet dans un fichier XML. C'est utiiser pour de la persistance, mais aussi pour des transactions (en particulier entre machines distantes dans une application JEE).

Tests Unitaires : Il s'agit de petits tests fonctionnels associés à chaque méthodes que vous codez qui peuvent se lancer aisément et souvent afin de se rendre compte rapidement que l'on a cassé un invariant.

TDD (hors programe) **:** *Test Driven Developement.* Une méthode managerial de développement consistant à écrire les tests en premier puis à coder en essayant de vérifier les tests un à un. Dans les faits, cela permet une interaction plus "dynamique" et plus "ludique", mais surtout, oblige à faire des tests...

Webservice (hors programe) **:** Interface machine-machine. Il s'agit d'un terme générique pour désigner la classe/fonction de votre programme ouverte sur le web et qui a pour but d'intéragir, non pas avec un humain, mais un autre programme via un protocole spécifique. Une utilisation très courante est pour l'interaction entre un backend sur un serveur (en Java par exemple) et une page dynamique (JavaScript) lancée sur votre navigateur.

2 Technologies spécifiques :

Derby : Base de données relationnelle utilisée en cours.

Docker : Logiciel de virtualisation légère utilisé pour le déploiement d'applications. cf. cours Docker (cours n.2)

Docker-Compose : Builder de Docker : permet de lancer plusieurs containers en parallèle et leur réseaux.

Eclipse (hors programe) **:** L'IDE Java le plus utilisé, mais en perte de vitesse (du moins dans les stages).

EJB : *Enterprise Java Bean.* Ce sont les classes embarquées dans l'application pour encoder la partie métier. cf. cours EJB (cours n.3)

Git : Il s'agit du gestionnaire de versions le plus utilisé dans l'entreprise. cf. cours git (cours n. 3)

Glassfish : C'est un des serveurs utilisés pour le cours. Il s'agit du serveur JEE distribué par Oracle, ce qui veut dire

qu'il utilise les normes de JEE par défaut.

Gradle (hors programe) **:** C'est un builder beaucoup utilisé.

Hibernate : C'est l'ORM historique. Il est encore beaucoup utilisé, soit directement (dans un projet Java non JEE), soit comme implémentation de la norme JPA.

HQL (hors programe) **:** *Hibernate Query Langage.* Langage de requêtes de Hibernate. Très semblable au JPQL.

IntelliJ : C'est L'IDE java utilisé en cours.

JavaEL (hors programe) **:**

JavaScript (hors programe) **:**

J2EE : synonyme de JEE

JEE : *Enterprise Java Edition.* C'est une suite de normes pour des frameworks standards (avec potentiellement plusieurs implémentations) comprenant, notamment, les normes JPA, EJB mais aussi une utilisation systématique du JNDI et de JDBC.

JPQL : *Java Persistance Query Langage.* C'est le langage de requêtes de JPA. Il s'agit d'un langage de requêtes objets qui s'adresse aux entités comme si elles étaient des classes d'une base de données objet virtuelle. L'ORM (ici JPA) se charge de transformer ces requêtes en SQL pour la base de données relationnelle utilisée.

JSF (hors programe) **:** C'est la technologie frontend de Java préconisée par Oracle. Malheureusement, on n'a pas le temps de l'aborder en cours car elle est beaucoup plus complexe que JSP.

JSON (hors programe) **:** *JavaScript Object Notation.* Remplaçant du XML, plus lisible et un peu plus puissant.

JSP : *Java Server Pages.* Il s'agit de la technologie de scriptlets utilisée par Java. Un .jsp est un fichier html avec des incrustations de commandes Java dans des balises `<% ---- %>` ou `<c: --- >`. L'idée est de générer une page fixe html dont le contenu a été généré par ces scriplets Java. Une page JSP est généralement acompagnée d'un ou plusieurs servlets pour gérer les requêtes de l'utilisateur et pour précompiler les données. Une page JSP est compilée vers une servlet et par la suite cette servlet génère la page html à l'ancienne. A noter que JPA tend à disparaitre au profit de JSF et des webservices, une techno frontend plus moderne, mais aussi plus complexe.

JUnit : C'est la librairie que l'on utilise pour les tests unitaires.

Kubernate (hors programe) **:**

Log4J : C'est la librairie de log utilisée.

Maven (hors programe) **:** Il s'agit du builder le plus courant. Dans les TPs, on peut l'utiliser quelques fois pour récupérer des librairies.

Netbeans (hors programe) **:** IDE très utilisé pour Java. C'est l'IDE gratuit qui est officiellement recommandé par Oracle.

Scala (hors programe) **:** C'est le principal langage fonctionnel issu de java. Il subit une croissance très forte ces dernières années.

Scrum (hors programe) **:** Scrum est décrit comme un "Framework" de développement agile ; par contre, il ne s'agit pas d'un framework logiciel mais d'un framework managerial (le mot est surchargé), qui n'entre pas dans le cadre du cours. Un intervenant extérieur devrait par contre venir vous en parler cette année.

Servlet : Techno historique de java pour le frontend. Il s'agit d'une classe implémentant des méthodes de réponses aux requêtes web. En général, on parle de servlet pour parler de httpservlet qui répond à des requêtes http. Les technologies JPA ou Spring MVC utilisent encore des servlet.

SpEL (hors programe) **:**

Spring : Il s'agit de l'un des plus vieux (et plus gros) frameworks. Il contient plusieurs modules, pour des tâches indépendantes. Spring met en avant l'idée IoC et permet à l'utilisateur d'en définir lui même.

SQL : *Standard Querry Langage.* Principal langage de requêtes pour les bases de données relationnelles

Tomcat : C'est le serveur Java le plus diffusé pour faire des application non-JEE de part sa légèreté. Il ne s'agit pas d'un serveur JEE, il ne peut donc pas déployer des EJB.

TomEE : C'est une extension (très récente) de Tomcat pour JEE.

Wildfly : C'est le serveur utilisé pour le cours. Il s'agit du serveur JEE de JBoss. On l'utilise car il fournit une image Docker officielle récente. Attention il gère différemment les connections à la BdD (voir le TP associé).

YAML (hors programe) **:** Format de fichiers de configuration moderne. Compatible avec JSON, il est plus lisible pour l'humain, mais moins adapté pour de l'interfaçage entre programmes (car moins redondant).

3 Jargon Docker :

Container Docker : Un conteneur Docker est un programme virtualisé par Docker. Il est séparé du reste des pro-

grammes tournant sur la machine au niveau système mais peut interagir avec elle selon les paramètres entrés. [3] Il est possible de limiter un conteneur en ressources (RAM, CPU, BP...). Un conteneur fait tourner une image Docker chargée dans la machine. Une seule image peut être utilisée pour plusieurs conteneurs.

Image Docker : Une image Docker est le programme que l'on fait tourner sur un (ou plusieurs) conteneur. Une image ne contient pas que le programme, mais aussi ses librairies et tout l'environnement associé, y compris le langage de programmation. Avec une seule image on peut faire tourner plusieurs conteneurs.

Mount Docker : Il y a deux moyens historiques de monter de la ROM sous docker (plus un plus récent avec les volumes). On peut monter un dossier de notre système (*bind mount*), mais il y a alors de gros risque de sécurité car une application dans un conteneur peut alors toucher à la machine (En général il vaut mieux utiliser les volumes dans ce cas). L'autre moyen (*tmpfs mount*) est de simuler de la mémoire morte en mémoire vive, par exemple pour Dockeriser une BdD dont la persistance n'est pas critique ou pour faire des tests. Dans le TP1_Docker, on la base de donnée est mise dans un *tmpfs mount*, elle disparait à l'arret du coneneur ; c'est bien pour les tests, mais pour un vrais serveur on utiliserait plutôt un volume Docker.

Nework Docker : Un réseau docker contient des container qu'il met en relation : un container `service` est ouvert aux autres container dans le même réseau Docker et accessible à l'adresse ip `service`. Ces connections sont internes et contenues : elles ne sont pas visible de l'extérieur du réseau, ne peuvent pas clasher avec d'autres services extérieurs et peuvent accéder à des ports inaccessibles depuis l'extérieur.

Volumes Docker : Un volume Docker est un fichier vide monté sur votre ordinateur au premier lancement d'une image, celuis-ci n'est pas détruit si le container est détruit et il sera utiliser par tous les conteneurs de la même image. C'est le moyen le plus safe de faire de la persistance sous Docker.

4 Jargon Spring :

Aspect (en AOP) : Par opposition au *main concern*, qui est le but principal de la méthode codé, un *aspect* est l'une des préoccupations secondaires que l'on doit traiter. La journalisation (ou *logging*, la sécurité, la gestion des exceptions, la gestion des transactions ou encore les tests, sont autant d'aspects possibles. Une même méthode peut embarquer plusieurs aspects.

Aspect AOP : Module de Spring en programmation par aspect.

Spring bean : L'injection de dépendance en Spring se fait en définissant des beans pour créer les objets dont on est dépendant. Il y a deux types de beans : des méthodes simples et des ensembles constructeur+setteurs ; dans le second cas, on parle de configuration.

Spring boot (hors programe) **:** C'est un des module de Sring. Il permet de récupérer une application complete mais sans contenu que l'on doit juste peupler. Attention, même si c'est très pratique pour ne pas avoir à refaire la même chose encore et encore, il faut tout de même comprendre ce que l'on fait et maitriser la structure du programme "vide".

Aspect MVC : Module de Spring pour le frontend (tier présentation), s'articule en model-vu-controleur.

5 Jargon JPA (et ORM en général) :

Cycle de vie : Une entité a un cycle de vie, celà veut dire qu'elle peut être dans plusieurs états : uniquement local, avec des modification local, iddentique a la version distante, existant en distant mais pas en local, etc... L'ORM est capable de manipuler corectement ces diférents états de façon invisible pour le programmeur.

Eager loading : C'est l'inverse de Lazy loading.

Entités : Lorsque l'on utilise JPA, une entité est une classe de la base de données virtuelle en paradigme objet. On peut récupérer un objet entité de cette base de données objet, en rechercher, modifier, ou même en stoker. Bien sûr, cette base de données étant virtuelle, il y derrière tout un mécanisme pour transformer les requêtes.

Lazy loading : cf. Lazy. C'est le mode par défaut de la plus part des ORMs : les données d'une entités sont récupérées que lorsque l'on accède à ses champs. Celà permet de ne pas chaîner innutilement les appels à la BdD pour des informations innutiles, maisça peut aussi multiplier innutilement les requètes.

Many-to-One : Il s'agit de lier, dans la BdD relationelle, les lignes d'une table t1 avec 0, 1 ou plusieurs lignes diférentes d'une autre t2 en ajoutant une columne t1_id dans t2. L'ORM est capable d'utiliser ce lien Many-to-one pour associer une entité t1 comme parametre des entités t2 et une liste d'entités t2 comme parametre des entités t1.

3. Attention : utiliser des conteneurs est simple lorsque l'on a pas de soucis de sécurité, mais s'ils font tourner des image sur lesquelles vous n'avez aucune maîtrise, il faut les paramétrer correctement !

Many-to-Many : Il s'agit de lier, dans la BdD relationelle, chaque ligne d'une table t1 avec 0, une ou plusieurs lignes d'une autre table t2, sans restriction sur le nombre de fois qu'une ligne de t2 est liée. Celà se fait en ajoutant une nouvelle table avec trois columns id, t1_id et t2_id qui représente une relation entre t1 et t2. L'ORM est capable d'utiliser ce lien Many-to-one pour associer une liste d'entités t1 comme parametre des entités t2 et une liste d'entités t2 comme parametre des entités t1.

One-To-Many : cf. Many-to-One

Unité de persistance : Il s'agit de la configuration de votre ORM. Elle contient les infos de connection de votre BDD et tous les paramètres sur la nature des liens entités-tables. Il y a beaucoup de moyens différents de la définir : unique fichier de configuration, références aux annotations, fichier de config + annotations, objet Java créé dynamiquement, infos incluses dans un fichier plus gros (comme le contexte d'application de Spring...)

6 Design patterns :

Application layer/tier : Tier/layer (cf. N-tier). La couche applicative (aka. de service) s'occupe de réorganiser les données calculées par la couche buisness afin que le la couche présentation ne face que de la mise en forme "esthétique". Concrètement, chaque présentation (JSP, servlet) n'a accès qu'a une classe de service, et dès que le codeur de la couche présentation a besoin de récupérer quelque-chose, il demandera au codeur du service de coder une méthode spécifique qui utilise les méthodes générales de la couche métier. Dans le TP1, les couches métier et application/services sont confondues (classes BiduleService) car l'application n'a pas vraiment de métier...

Buisness layer/tier : Tier/layer (cf. N-tier). La couche métier est celle qui fait les calculs/réflexions un peu complexe, qui factorise les opérations utilisées par plusieurs services, et qui récupère les informations de différents accesseurs (cf. data acces layer). Dans le TP1, il n'y en a pas encore (on se contente d'un service).

Data acces layer/tier : Tier/layer (cf. N-tier). La couche accesseur regroupe les classes qui permettent d'accéder aux fournisseurs (bases de donnée, les loggeurs, et autres services sous-traitants).

DAO (hors programe) **:** *Data Acces Object*. Étant donnée une entité ou une interface (ou classe abstraite) sur des entités `PizzaEntity`, il est généralement conseillé d'écrire un `PizzaDAO`. Il s'agit d'une classe pouvant récupérer, modifier ou enregistrer des objets de l'entité. Dans une application N-tier classique, il se place entre le service et l'entité. Un service est lié à l'utilisation et un DAO est lié à la donnée. [4]

Factory : Techniquement, une `FooFactory` est une classe dont les objets disposent de méthodes `Foo createFoo(...)` permettant de construire des `Foo`. Dans la pratique, cela a deux intérêts. D'abord, cela permet de fournir des constructeurs pour des classes abstraites et des interfaces (avoir une classe séparée permet alors de savoir quelles implémentations sont disponibles...). Mais la factory a aussi un intérêt plus subtil : ça permet l'instanciation partielle ; [5] si l'on veut construire plusieurs objets sous un même schéma mais avec de petites différences subtiles, on instancie une factory avec les paramètres schéma général et on utilise la méthode de création avec un petit paramètre servant à fixer les petites spécificités.

Layered : Voir N-tier.

Microservices : Il s'agit de concevoir les composantes de l'application indépendamment et avec une interaction réseau. Cela scale mieux sur un cluster/cloud si vous n'utilisez pas de serveur distribué. Par contre, cela introduit une latence non négligeable sur l'interaction entre les microservices, latence qu'essaient de réduire des technologies comme Docker.

MVC : *Model View Controler*. Il s'agit d'un schéma d'architecture classique très utilisé pour les applications à composante graphique. L'idée est que l'utilisateur ne voit que la vue et ne s'adresse qu'au contrôleur, tandis que le contrôleur va pouvoir instancier le modèle ce qui va modifier la vue.

N-tier : Schéma d'architecture où les composantes (appelées tiers) sont rangées en chaîne, le premier tier interagissant avec le client et chaque tier n'interagissant qu'avec ceux qui lui sont adjacents. [6] On parle de 2-tier, 3-tier... Par exemple, Frontend-Backend est une architecture 2-tier, Presentation-Application-Data est une 3-tier et Presentation-Service-Buisness-Persistance est une 4-tier. Attention, la nomenclature n'est pas bien définie, et certaines personnes considèrent que l'on appel "tier" des composantes distantes (browser-serveur-BdD) et "layer" des composantes sur une même application, dans ce cas on parle d'architecture "layered"

Presentation layer/tier : Premier tier d'une architecture N-tier. Il s'occupe de la mise en forme pour interagir avec le client. Dans JEE il s'agit des JSPs et servlets.

Persistence layer/tier : Tier/layer (cf. N-tier). La couche de persistance est une spécification de la couche accesseur

4. Dans le TP1, on fusionne les deux dans MenuService car l'application est très simple, l'une des étapes suivantes de complexification de l'apli serait de faire cette séparation.

5. Tend à disparaître avec l'apparition des lambda-expressions dans Java8.

6. Lorsqu'il y a beaucoup de tiers, on peut des fois accepter une interaction secondaire à une distance de 2...

(cf. data access) pour les bases de données. Attention, on parle là de l'accès aux bases de données, et non pas du mot général Persistance qui désigne toute méthode de stockage. Dans JEE, les entités représentent la couche persistance.

POJO (hors programe) **:** *Plain Old Java Object.* Il s'agit d'un nom fancy pour parler d'un objet java normal. Ce nom a été donné ironiquement pour critiquer l'utilisation systématique de bidule beans ou machin servlet... Le seul fait qu'il soit resté dans le langage comme un terme technique en dit long sur la communauté Java...

Service layer/tier : voire Application layer.

VO (hors programe) **:**

7 APIs et classes Java :

EJB CMT (hors programe) **:** . C

Entity Manager : Il s'agit de l'interface qui vous connecte à l'ORM utilisé. Elle est construite à partir d'une unité de persistance.

JDBC (hors programe) **:** *Java DataBase Connectivity.* C'est l'API qui gère les ouvertures/fermetures de connections aux bases de données. Dans le TP, JDBC est gérée par JPA (TP1) ou Spring (TP2).

JNDI (hors programe) **:** *Java Naming and Directory Interface.* Il s'agit du dictionnaire des noms et labels apparaissant dans une appli JEE. Il contient le descriptif de la base de données, mais aussi tous les noms/chemins des fichiers de configuration.

JPA : *Java Persistance API.*[7] Il s'agit de la norme (et de l'implémentation référente) d'ORM dans JEE.

JTA (hors programe) **:** *Java Transaction API.* C'est l'API JEE générale pour gérer tout types de transactions : avec les BDD, *queues...*, mais aussi entres EJBs. JPA implémente une spécification de JTA pour une BDD relationnelle (mais est plus que cela). Il s'agit de la norme des interactions entre EJBs (possiblement distant) dans JEE. Elle est toujours présente dans les applications compilés, mais vous ne la voyez pas directement à moins de faire du thuning précis du déploiement. Dans ce dernier cas, on peut aussi utiliser des EJBCMT (*EJB Container Managed Transactions*) qui sont prennent la forme d'annotations plutôt que de classes de messages.

λ-objets/expressions (hors programe) **:** C'est un des apports de Java 8. Il s'agit d'objets représentant une unique méthode, permettant d'utiliser des primitives de programmation fonctionnelle.

Serializable : Cf. Serialization. C'est l'interface des classes sur lesquelles on peut faire de la sérialisation. Elle ne nécéssite pas de méthode, mais tous les champs doivent être eux-même sérialisable (de rares objets ne sont pas sérialisable comme les λ-objets). Un EJB est toujours sérialisable car c'est le seul moyen de le passer en argument à une méthode disponible sur une machine distante.

8 Annotations :

@AttributeOverride(s) :
@Autowired :
@Basic :
@Bean :
@Column :
@ComponentScan :
@Configuration :
@EJB :
@Embeded :
@Entity :
@Id :
@ManyToMany :
@ManyToOne :
@MappedSuperclass :
@NamedQuery :
@OneToMany :
@Override :
@PersistenceContext :
@PersistenceUnit :

7. Oui, c'est bien un acronyme d'acronyme...

@Primary :
@RequestMapping("toto") :
@Scope :
@Stateless :
@Table :

9 Extensions de fichiers :

.class (hors programe) **:** Fichier compilé de java (utilisé par la JVM).
.form (hors programe) **:** Extension qui n'existe pas. On l'utilise comme extension virtuelle dans Spring MVC. cf TP.2
.jar : Archive java. Utilisée pour archiver les librairies java.
.java : Fichier de code java
.json (hors programe) **:** Fichier de config JSON. cf JSON
.jsp : Extension de JSP
.pom (hors programe) **:** Extension de Maven
.war : Archive java. Utilisée pour les applications web javaEE. Peut être exécutée par un serveur javaEE (comme glassfish).
.xml : Fichier de config xml (utilisé par beaucoup de frameworks).
.yaml (hors programe) **:** Fichier de config YAML. cf YAML
.yml (hors programe) **:** synonyme de .yaml