

Welcome to AMT 2020

Week 2

- **Objectives**

- Be able to explain what it means for a component to be "managed" and by "what" components are managed.
- Be able to explain what dependency injection means in object-oriented programming.
- Be able to explain how dependency injection can be implemented "from scratch". See <https://github.com/SoftEng-HEIGVD/Teaching-HEIGVD-AMT-Dependency-Injection>
- Be able to explain how dependency injection is implemented in the Java EE / Jakarta EE platform.
- Be able to explain what Aspect Oriented Programming (AOP) is.
- Be able to explain the relationship between AOP, managed components and dependency injection.

- **Project**

- Replace Payara / Wildfly with Open Liberty, a modern implementation of Jakarta EE (see [this webcast](#))
- Study the reference architecture described in [this](#) and [this](#) webcast, which provides the overall structure for your application across all tiers.
- Reuse the code you wrote during the first week.
- Make sure that your CodeceptJS tests still work and extend them.
- Implement a new feature (users can ask questions)
- Implement a first continuous integration pipeline with GitHub Actions (build and publish a Docker image as described in [this webcast](#)).

- **The Question of the Week:**

- *Expliquez comment les Enterprise Java Beans (EJBs) permettent d'appliquer les principes de la programmation orientée aspects (AOP). Appuyez-vous sur un exemple de code et expliquez en détails ce qui se passe "derrière les décors" (i.e. expliquez ce que le serveur d'application fait).*

Context : beaucoup de plate-formes s'appuient sur la notion de "composants managés" pour faciliter la vies des développeurs. Puisqu'elles contrôlent le cycles de vie des composants, elles peuvent injecter des comportement dans différentes situations (sécu, gestion des transactions...)

Définitions :

- AOP : but -> séparation des préoccupations, code métier vs code technique, ex -> logging, sécu.
Perspective historique ex de code sans AOP -> pd à résoudre
- EJB : spécification que fait partie de JavaEE, API qui permet de créer des composants managés par le serveur d'application

<h2>Introduire le contexte</h2>	<p>Beaucoup de plate-formes s'appuient sur la notion de "composant managés" pour faciliter la vie des développeurs. Puisqu'elles contrôlent le cycle de vie des composants, elles peuvent injecter des comportements dans différentes situations (sécurité, gestion des transactions, etc)</p>
<h2>Poser les définitions</h2>	<p>AOP</p> <ul style="list-style-type: none"> + but: séparation des préoccupations + code métier vs code technique + exemples: logging, sécurité + perspective historique + exemple de code SANS AOP (problème à résoudre) <p>EJB</p> <ul style="list-style-type: none"> + spécification qui fait partie de Java EE + API qui permet de créer des composants managés par le serveur d'application
<h2>Répondre à la question</h2>	<p>Code</p> <ul style="list-style-type: none"> + stateless session bean avec une méthode + servlet, injection de dépendance et appel de la méthode <p>Schéma</p> <ul style="list-style-type: none"> + schéma présenté au cours, avec le proxy généré
<h2>Exposer ses arguments</h2>	<p>Explications</p> <ul style="list-style-type: none"> + ce qui se passe au déploiement (serveur d'app scanne le code et traite les annotations). + pour chaque EJB, il génère un proxy et il injecte une référence vers ce proxy dans le servlet + le servlet n'a donc pas une référence directe sur une instance de la classe codée par le développeur + lors de l'appel de la méthode, le code du proxy est exécuté (ce qui permet par exemple de démarrer une transaction). Le proxy fait finalement un appel sur le code "métier" écrit par le développeur.
<h2>Conclure</h2> <p>Il y a d'autres moyens de faire de l'AOP en Java / Java EE, notamment avec le framework Spring.</p>	

- **Notes about the YouTube Playlist:**

- I have added 4 videos (see the links in the Project paragraph just above).