

# Conception d'applications web en Java

## Présentation du cours

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 26 septembre 2016

# L'enseignant

- Olivier Cailloux
- [olivier.cailloux@dauphine.fr](mailto:olivier.cailloux@dauphine.fr)
- Coordonnées : cf. [annuaire](#) de Dauphine

# Objectifs pédagogiques

- Programmer des applications d'envergure
- De qualité
- Portables

Complexité :


- Appui sur API ouvertes
- Programmer selon les specs
- Beaucoup de paramètres  $\Rightarrow$  comprendre !

# Un environnement imparfait

Log Viewer – Mozilla Firefox

SP servlet best pract... x Maven – Introduc... x http://loc...ityManager x PersistenceCont... x PersistenceCont... x hibernate - When... x

localhost:4848/common/logViewer/logViewer.jsf?instanceName=server&loglevel=INFO&viewResults=true

 **An error has occurred**  
Check server log for more information.

## Log Viewer

View, search, and filter a server log file using basic and advanced options. Refer to the Log Levels page for information about log levels you can filter here.

[Advanced Search](#)

### Search Criteria

**Text search:**

Only log entries containing the specified text will be displayed. Search is case sensitive.

**Timestamp:** ☒ **Most Recent** ☐ **Specific Range:**

**Log Level:**  ☐ **Do not include more severe messages**

Log entries are limited to those stored in the log file. Set appropriate log level in the Log Level page to ensure data is logged.

[Modify Search](#)

**Instance:**

**Log File:**

**Log Viewer Results (0)**

# Un environnement complexe

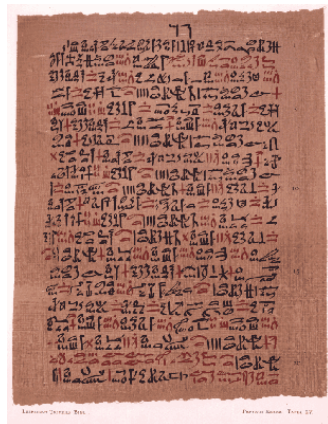
De <http://graphserver.github.io/graphserver/> :

*First, get Java 1.6. Or Java6. I think they're the same thing. Get the JDK, but not the JVM, the JRE, SDK, SDN, or the JCE. Note Java SE comes with Java EE, which is apparently at version 5, and may or may not have anything to do with J2EE. I don't know what those have to do with anything. Google it or something. I don't know. They don't make it particularly easy for you. It's like, they've got more money than god and nothing pleases them better than pouring all that expertise into baffling the hell out of you. Honestly, I can't stand Java, but you need it to run Osmosis.*

La complexité est-elle une mauvaise chose ?

# Les bienfaits de la complexité

- Activité plus difficile : souvent plus attrayante
- Évite les activités répétitives : complexité amène diversité
- Récompenses plus grandes
- Recherche d'un accomplissement personnel
- Cercle vertueux : accès à activités plus complexes
- Pas un bien positionnel : accessible à tous



Écriture **hiératique**, égypte ancienne

# Objectifs pédagogiques

## Mise en œuvre des patterns

- Mise en œuvre des patterns dans les spécifications Java EE
  - Quand les mettre en œuvre ?
  - Applications dans programmes propres
- 
- Prise en main d'outils de dév avancés :
    - Eclipse ;
    - Maven ;
    - Git...

# Objectifs pédagogiques, suite

## Modélisation

- Réponse à des besoins exprimés vaguement
- Appui sur les standards du web actuels
- Réusinage fréquent
- Dosage du réalisme et de l'intérêt des fonctionnalités

Approche agile ? À moitié !

- Livraisons fréquentes
- Travail en binôme
- Réusinage intense



# Intérêt pratique

- Qu'on soit programmeur, qu'on discute avec des programmeurs
- Prendre de la hauteur, éviter les tâches répétitives et se concentrer sur le conceptuel
- Respect et compréhension des standards (aperçu de la façon dont ils sont construits) : compétence essentielle
- ... dans de multiples domaines
- Importance des patterns dans de multiples domaines
- Technologie en vogue
  - Java EE VS Spring VS ... ?

# Implémentation

- Implémentation des techniques sur un projet
- Objectif : un projet utile
- Un projet  $\neq$  par groupe
- Programmation en binôme, équipes tournantes :  
diversification, pas spécialisation
- À gérer de façon agile car fonctionnalités ajoutées au fil du  
cours : cycles courts, réusinage...
- Fonctionnalités de l'application décrites de manière vague : à  
vous de m'interroger
- Fin d'année : présentation collective de vos projets
- Vote pour la meilleure application
- Livraisons exclusivement via Git

# Évaluation

L'évaluation se fait par contrôle continu exclusivement :

- qualité du code
- respect des demandes de l'utilisateur
- mise en œuvre adéquate des technologies dans l'application
- livraisons régulières
- ampleur des fonctionnalités
- qualité générale de l'application
- qualité de la présentation finale...
- Note finale : agrégation des notes reçues au long de l'année (nombre de notes : aléatoire)
- Vous êtes toujours censés comprendre votre code

# Prérequis

- Programmation en Java, manipulation d'un environnement de développement, compréhension des notions algorithmiques élémentaires.
- Capacité à comprendre des textes en anglais liés à l'informatique.
- HTTP, HTML, XML, XSD, SQL.

# Mise en œuvre

- Pédagogie par projet et *partiellement* inversée
- Central : travail sur projet
- Durant cours  $x$  : constituez un binôme  $\neq$  du précédent
- Travaillez sur une fonctionnalité de votre projet
- Doit mettre en œuvre la techno vue au cours  $x$
- À remettre avant cours  $x + 1$
- Début cours  $x + 1$  : je tire des personnes au hasard
- J'évalue, avec le binôme, le travail fourni depuis la dernière évaluation
- Lors premiers cours : une brève introduction à une technologie par cours
- Je joue le rôle du client : détail des fonctionnalités à implémenter

# Plan

- Cours 1 : Git, aperçu de Java EE, Maven
- À faire : lire doc eclipse, canevas projet de groupe + branche personnelle, servlet simple
- Cours 2 : CDI.
- À faire : modèle singleton (injecté), lié à servlet.
- Cours 3 : JPA.
- À faire : modèle persistant.
- Cours 4 : JSF.
- À faire : pages JSF.
- Puis, approfondissements et élaboration du projet

Plus : Éléments d'ingénierie (programmation par contrat...)

# Astuces importantes

- Éviter l'essai / erreur : comprendre
- Poser des questions !
- Commencer simple
- Si ça ne fonctionne pas : faire plus simple
- Exclure bugs possibles pas à pas
- Éviter débuggage pas à pas
- Se méfier du code auto-généré
- Choisir *une* approche cohérente. Mélange deux tutos : problèmes presque assurés ( $\neq$  versions ;  $\neq$  annotations...)

# À faire

Ce cours :

- [Choix](#) d'un projet
- Indication du choix sur [grille de suivi d'avancement MyCourse](#)
- Nom projet sur MyCourse = nom technique

Avant le prochain cours :

- Rediriger vos e-mails @ Dauphine si nécessaire pour vous assurer de recevoir les annonces
- Installer les outils sur votre machine : Eclipse Neon Java EE, Java EE 7 (et Glassfish 4), Java 8 (OpenJDK), git
- Compte [GitHub](#)
- Commit initial projet Git de groupe
- Transformez l'indication dans la grille de suivi en un lien vers l'url Git de votre projet



# À faire

Avant chaque cours :

- Constituez un binôme  $\neq$  du précédent
- Par binôme : Choix d'une issue sur projet Git (création éventuelle)
- Par binôme : Commit branche perso, nom branche = n° issue
- Par chacun : Indication sur MyCourse (dans devoirs) de l'URL issue travaillée + nom binôme
- Conseil : *vérifier* que je verrai votre code en effectuant un nouveau clone vous-même

# Sondage !

- Avez-vous une machine à amener en cours ?
- Quels sont les prérequis qui vous manquent ?
- Est-ce une bonne idée d'effectuer les prochains cours en salle de classe (sans machines, mais avec prises de courant) ?
- Autres remarques sur l'approche proposée ? (Contenu du cours ; craintes ; envies...)

# Licence

Cette présentation, et le code LaTeX associé, sont sous [licence MIT](#). Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.  
Le travail réutilisé est à attribuer à [Olivier Cailloux](#), Université Paris-Dauphine.