Conception d'applications internet

CDI

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

Était à faire

Exercice à effectuer *avant* le 25 janvier, par chaque membre de chaque équipe.

+ [1] Un servlet qui applique une transformation quelconque sur un attribut d'un ou plusieurs objets stocké dans votre BD. Cf. Exercices.

Exercices :

- Programmer une méthode qui transforme un attribut d'un objet. Par exemple, elle met le nom en majuscule s'il ne l'était pas (obligation d'utiliser Java, pas SQL : supposez que la transformation est trop complexe pour être exprimée en SQL).
+ Permettre l'application de cette méthode via un servlet. Votre servlet ne doit pas nécessairement accepter de paramètres. Attention à l'atomicité de la transaction !

1. Le + indique que cet aspect intervient dans la note

Dépouillement

- En résumé : détails demandés ; meilleur lien exercices et projet
- En contrepartie : couverture restreinte

Plan du reste du cours

- Cours 5, 6 : CDI (et retour sur EJBs)
- Cours 7, 8 : JSF
- Cours 8 : présentation des projets + vote meilleur projet

Également disponible sur MyCourse : JPA ; JAX-RS (services web REST)

Modification évaluation

- Fin des exercices notés
- Note actuelle, si positive, comptera dans pondération individuelle
- Note concept évaluée selon code du projet (sauf demande ≠)

Note projet globale

Technologies vues : {EJB, Servlet, JDBC ou JPA, CDI, JSF}

Note projet globale

- Exactitude ; maintenabilité du code ; fonctionnalités
- Utilisation appropriée des technologies vues
- Suivre recommandations données au cours (ou en discuter)
- Si divergences (par ex. JSP au lieu de JSF) : me demander

Pondération individuelle

Pondération individuelle

- Éviter la spécialisation : chaque membre doit utiliser directement au moins trois techno. vues, auteur ou reviewer
- Pondération si déséquilibre de qual/quantité de contribution
- Pair programming apprécié
- Contribution significative en tant qu'auteur demandée

Chaque commit doit indiquer clairement l'auteur et éventuellement le reviewer

- Utilisez `user.name`, un seul nom, gardez le même nom
- Indiquez dans le commentaire du commit le binôme éventuel : « reviewer: Machin » avec Machin le user.name du binôme

Utilité de l'injection

CDI ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur…)

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur...)
- MAIS crée une dépendance encombrante
- Et si classe créée automatiquement ? (exemple ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur…)
- MAIS crée une dépendance encombrante
- Et si classe créée automatiquement ? (exemple ? Servlet !)
- Solution Java EE : CDI
- Requiert gestion du cycle de vie

Vue d'ensemble

- Bean : classe dont le cycle de vie des instances peut être géré par le conteneur (y compris ressources)
- Pour ce faire, préciser son « contextual scope » : `@RequestScoped` ; `@SessionScoped` ; `@ConversationScoped` ; `@ApplicationScoped`…
- Conteneur crée *instance contextuelle* à la demande
- Instance contextuelle détruite à la fin de son scope
- Injection : `@Inject` sur champ ou méthode ou constructeur

Context object

- *Context object* de scope S : associe un type de bean à maximum une instance contextuelle de scope S
- Un context object par requête / session / conversation / ...
- Injection possible seulement dans objet ayant accès au scope adéquat
- Recommandation : toujours préciser le scope pour l'injection

Scopes

- `@RequestScoped` : détruit après traitement de la requête
- `@SessionScoped` : référence liée à la session HTTP ; détruit avec la session (cf. Cours 3 EJB)
- `@ApplicationScoped` : créé au besoin, un par application
- `@ConversationScoped` : gestion manuelle de la conversation
- `@Dependent` : pas un normal scope ; scope lié à la destination de l'injection
- Session et conversation-scoped : implémenter `Serializable`

Conversations

- `@ConversationScoped` : gestion manuelle de la conversation
- `Conversation` : objet `@RequestScoped` (donc injectable) pour démarrer et terminer une conversation
- Une conversation a un identifiant unique
- Joindre une conversation : envoyer paramètre de requête `cid`

Scopes et EJB

- Stateless session bean doit avoir scope `@Dependent`
- Singleton doit avoir `@ApplicationScoped` ou `@Dependent`
- Stateful : tout est permis

Scopes et concurrence

- Le développeur doit protéger les objets dans un scope contre les accès concurrents
- Sauf ConversationScoped, garanti par le conteneur actif pour une requête à la fois

Désambiguation et production

Comment préciser quelle classe injecter ?

- Un bean a des *qualifieurs* : annotations spécialisant un type
- Qualifieur : type annotation annoté `@Retention RetentionPolicy.RUNTIME` et `@Qualifier`
- Préciser `@Inject @Qualifieur Type truc;`
- Qualifieurs prédéfinis : `@Default` (tout type sans qualifieur ou slmt `@Named`), `@Any` (tout sauf `@New`)

Producteurs

- Méthode `@Produces` : produit une instance à la demande
- Méthode peut aussi être annotée d'un scope et de qualifieurs (s'appliquent au produit)

Divers

- `@Transactional` démarcation automatique de transaction
- On peut injecter : managed beans et Java EE Resources
- Ressource : bean représentant référence vers environnement composant (`@EJB`, `@Resource`, @WebServiceRef, slmt scope `@Dependent` standard)

Tests

- Comment faciliter les tests unitaires (par exemple) ?

Divers

- `@Transactional` démarcation automatique de transaction
- On peut injecter : managed beans et Java EE Resources
- Ressource : bean représentant référence vers environnement composant (`@EJB`, `@Resource`, @WebServiceRef, slmt scope `@Dependent` standard)

Tests

- Comment faciliter les tests unitaires (par exemple) ?
- Ajouter des méthodes `setTruc(Truc)`, visibilité package
- Test sans CDI : initialiser la classe avec `new`

Références

- The Java EE Tutorial: Contexts and Dependency Injection
- JSR 346 (Context and Dependency Injection 1.1 et 1.2) (direct).
- JSR 330 (Dependency Injection) (direct) : simplement `@Inject` et cie
- JSR 330, section 4 : description courte de l'intérêt du DI
- JSR 342 (Java EE 7) (direct)
- JSR 345 (EJB 3.2) (direct)

Exercices I

- Programmer un bean `@RequestScoped`
- L'injecter et l'utiliser dans un servlet ou autre objet avec portée requête
- Programmer un bean `@RequestScoped` avec un état (un simple compteur, par exemple, qui s'incrémente lors de chaque utilisation)
- Observer son comportement lors de l'utilisation dans un servlet
- Rendre votre bean `@ApplicationScoped` ; observer ce que ça change
- Rendre votre bean `@SessionScoped` ; observer ce que ça change

Exercices II

* Simuler un traitement long (`Thread.sleep`) dans le bean CDI. Avec deux navigateurs différents, envoyer deux GET simultanées. On souhaite qu'elles soient traitées en parallèle (la deuxième n'attend pas la fin de la première avant de s'exécuter). Comment annoter le bean CDI pour ce faire ? Quelle différence prévoyez-vous par exemple entre `@ApplicationScoped` et `@RequestScoped` ?
* Pour le vérifier, modifiez le comportement de votre bean CDI. Il possède maintenant un compteur `i`. Il incrémente `i` en entrée d'appel, puis il renvoie la valeur de `i` à l'appelant, puis il décrémente `i`. Le bean renverra-t-il toujours la valeur d'initialisation de `i` + 1, lors d'appels parallèles de clients web différents ? Prédire son comportement en fonction de la portée CDI qui vous lui affectez. Réfléchir aux avantages de différents

Exercices III

choix. Pour vérifier vos prédictions, utiliser `Thread.sleep` judicieusement.

- Programmer un SLSB (stateless session bean), l'injecter avec `@EJB` : quelles différences avec un bean `@RequestScoped` ? Et quelles différences avec d'autres portées ?

Licence