Applications orientées données (NSY135)

11 – Le langage HQL

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

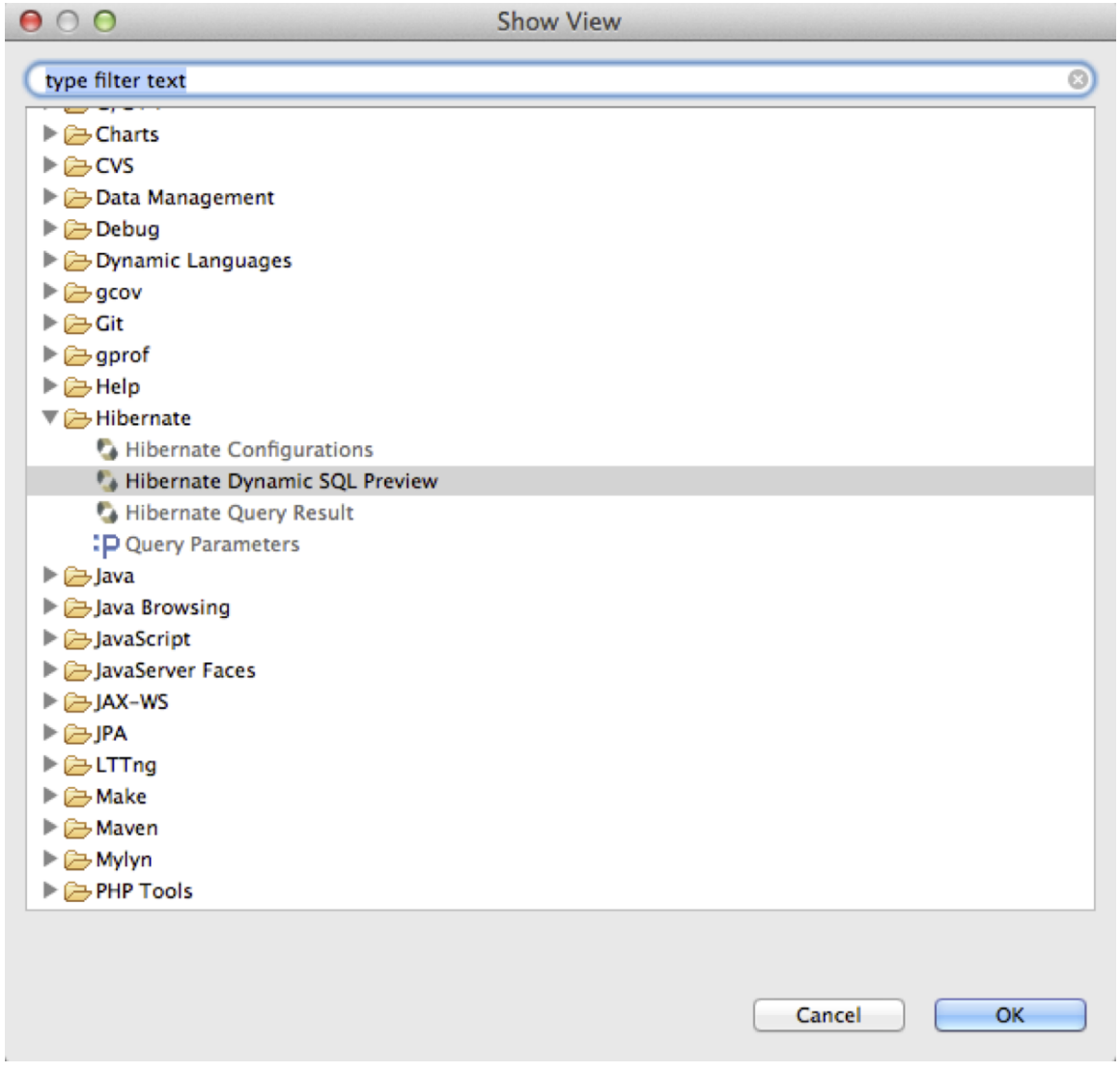
Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

Introduction

- On présente ici les aspects essentiels de HQL, à compléter par la documentation officielle au besoin
- installons sous Eclipse du plugin **Hibernate Tools**
  - Sous Eclipse, dans le menu **Help**, accédez au choix **Marketplace** (ou **Software search and updates**, selon les versions).
  - Lancez une recherche sur **Hibernate Tools**.
  - Lancez l'installation, puis relancez Eclipse quand elle est terminée.
  - Une fois le **plugin** installé, vous pouvez ouvrir des **vues** Eclipse.
  - Dans le menu **Windows**, puis **Show view**, choisissez l'option **Hibernate**

Hibertools


Show View
type filter text
Charts
CVS
Data Management
Debug
Dynamic Languages
gcov
Git
gprof
Help
Hibernate
Hibernate Configurations
Hibernate Dynamic SQL Preview
Hibernate Query Result
Query Parameters
Java
Java Browsing
JavaScript
JavaServer Faces
JAX-WS
JPA
LTTng
Make
Maven
Mylyn
PHP Tools
Cancel
OK

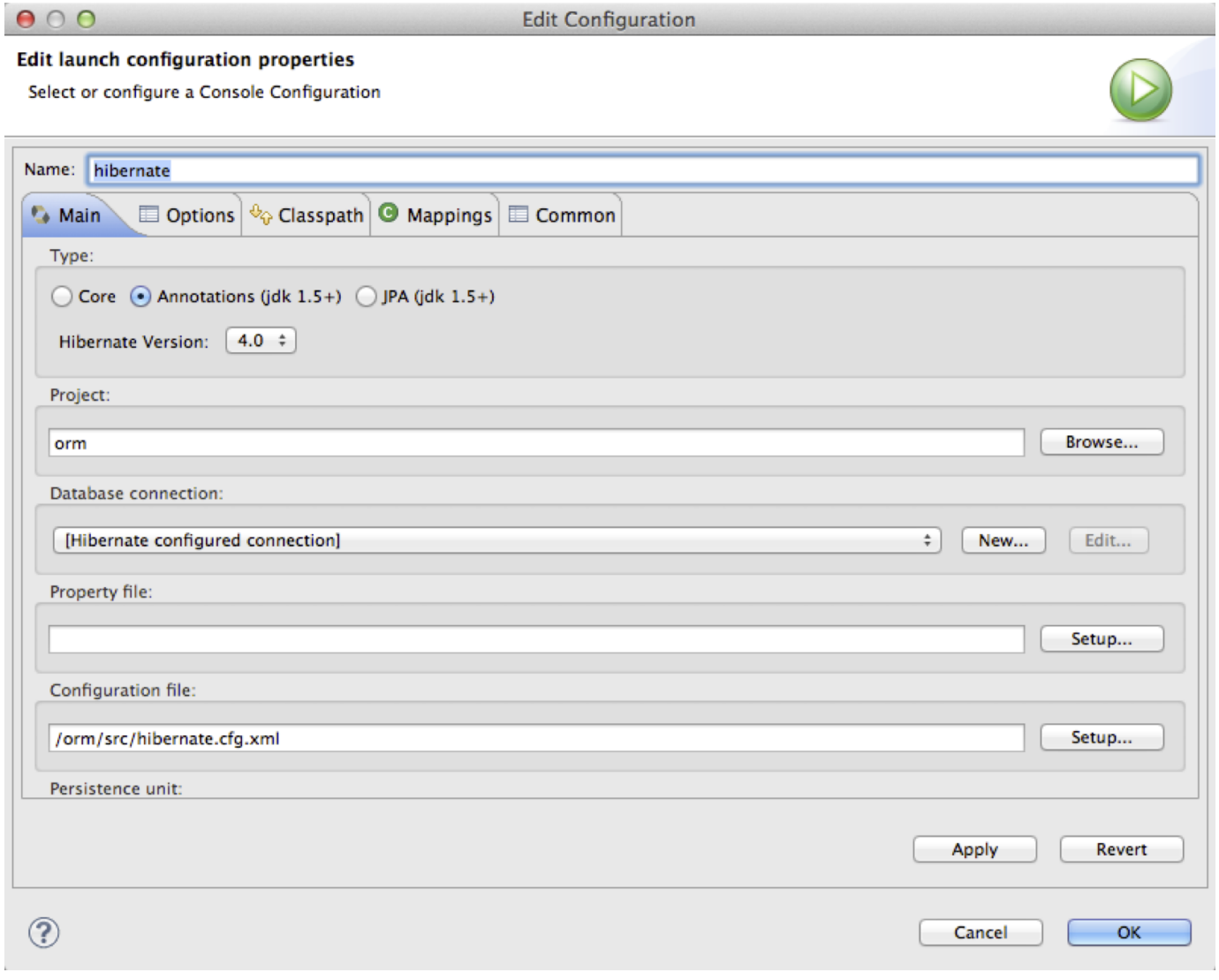
Configuration : guide rapide

- Lancez tout d'abord la fenêtre **Hibernate Configurations**.
- Vous pouvez créer des configurations (fichier **hibernate.cfg.xml**).
- Une configuration peut être créée à partir d'un projet existant, ce qui revient à utiliser les paramètres Hibernate déjà définis pour le projet.

- Quand vous êtes sur l'onglet **Configuration** de la nouvelle fenêtre, des boutons sur la droite permettent de créer une nouvelle configuration
- Pour une configuration existante, le bouton droit donne accès à l'édition de la configuration, au lancement d'un éditeur HQL, etc.

- Le plus simple est donc d'indiquer le projet associé à la configuration, et d'utiliser le fichier de **mapping** **hibernate.cfg.xml** de ce projet.
- Cela assure que les classes persistantes et la connexion à la base sont automatiquement détectées.
- L'outil vous permet également de créer ou modifier un fichier de configuration avec une interface graphique assez agréable.

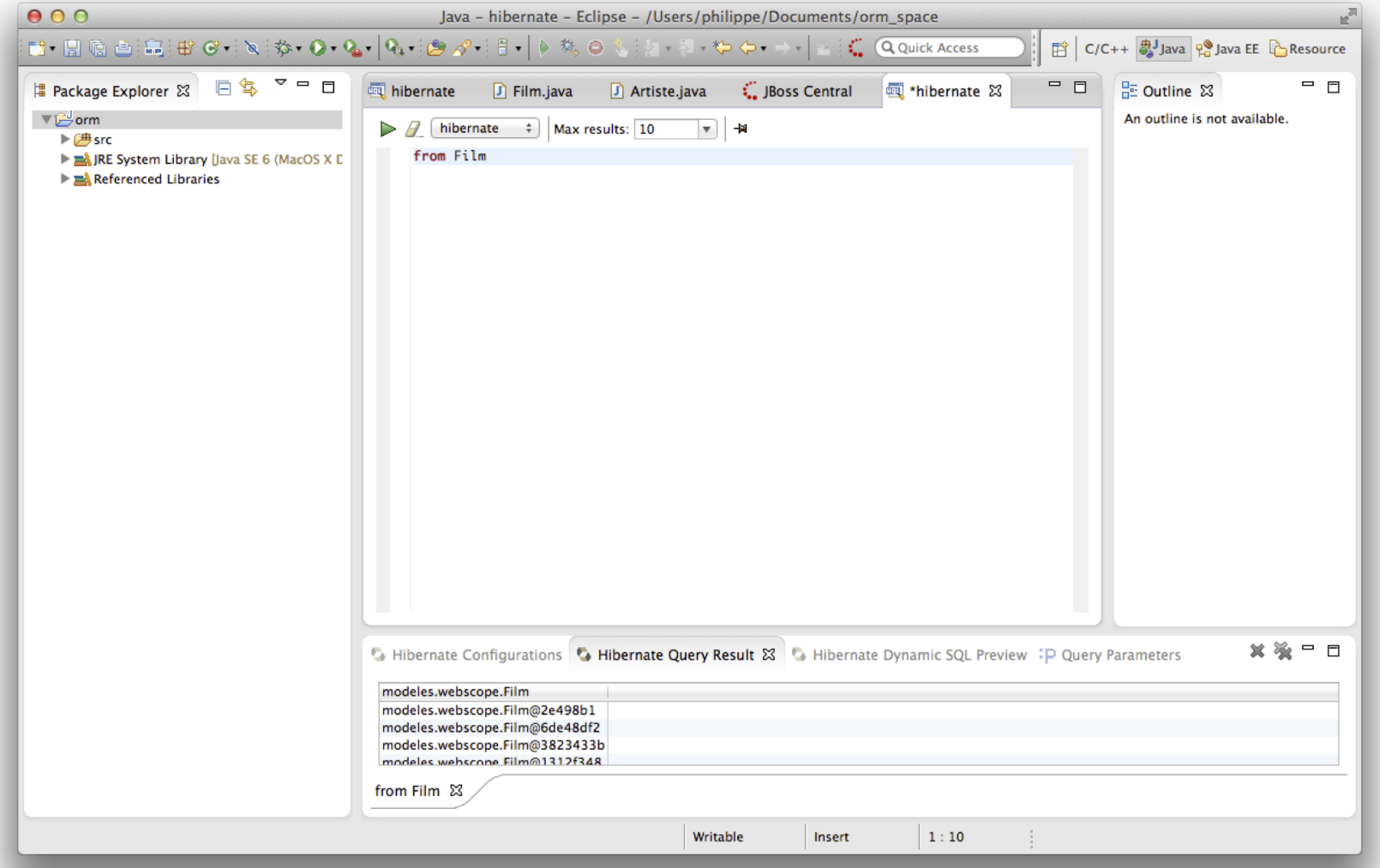
Configuration d'une session d'Hibernate



Utilisation

- Une fois la configuration créée, vous pouvez lancer un éditeur HQL et y saisir des requêtes.
- Pour l'éditeur HQL, sélectionnez votre configuration de session, et utilisez soit le bouton droit, soit l'icône "HQL" que vous voyez sur la barre d'outils.
- Vous pouvez saisir des requêtes HQL dans l'éditeur, et les exécuter avec la flèche verte.
- Trois autres vues sont utiles:
  - **Hibernate Dynamic SQL** Preview vous montre la/les requête(s) SQL lancée(s) par Hibernate pour évaluer la requête HQL (très utile pour comprendre comment Hibernate matérialise le graphe requis par une requête HQL)
  - **Hibernate Query Result** vous donne simplement le résultat de la requête HQL.
  - **Query parameters** sert à définir la valeur des paramètres pour les requêtes qui en comportent.

Les fenêtres Hibertools



Utilisation

- les requêtes HQL renvoient en général des **objets**
- Hibernate Tools affiche la **référence** de ces objets (peu parlant)
- Pour consulter plus facilement le résultat des requêtes, on peut ajouter une clause **select** (optionnelle en HQL) :

```
select film.titre, film.annee
from Film as film
```

HQL et Java

- L'intégration de HQL à Java se fait par des méthodes de la **Session**. Le code suivant recherche un (ou plusieurs) film(s) par leur titre.

```
public List<Film> parTitre(String titre)
{
Query q = session.createQuery("from Film f where f.titre= :titre");
q.setString ("titre", titre);
return q.list();
}
```

- La clause **select** est optionnelle en HQL
- elle offre peu d'intérêt dans une application java car on cherche en général à récupérer automatiquement des instances des classes **mappées** : cela implique de récupérer tous les attributs de chaque ligne.

HQL et Java (2)

- Commme en JDBC, on peut introduire dans la requête des **paramètres** en les préfixant par “:“ ("?" est également accepté).
- Hibernate se charge de protéger la syntaxe de la requête (en ajoutant des barres obliques devant les apostrophes et autres caractères réservés)
- Voici une requête un peu plus complète (et plus concise) :

```
session.createQuery("from Film as film where film.titre like :titre and film.annee < :an
.setString ("titre", "%er%")
.setInteger("annee", 2000)
.list();
```

- On applique la technique dite de **chaînage des méthodes** :
  - chaque méthode **set** renvoie l’objet-cible
  - objet auquel on peut donc appliquer une nouvelle méthode
  - et ainsi de suite
- Notez également que l’affectation des paramètres tient compte de leur type: Hibernate propose des **setDate()**, **setInteger()**, **setTimestamp()**, etc.

HQL et Java (3)

- On peut également utiliser comme paramètre un objet persistant, comme le montre l'exemple suivant:

```
// bergman est une instance de Artiste
Artiste bergman = ...;

session.createQuery("from Film as film where film.realisateur= :mes")
.setEntity ("mes", bergman)
.list();
```

- HQL offre la possibilité de **paginer** les résultats, une option souvent utile dans un contexte d'application web.
- L'exemple suivant retourne les lignes 10 à 19 du résultat.

```
session.createQuery("from Film")
.setFirstResult (10)
.setMaxResults(10)
.list();
```

HQL et Java (4)

- La méthode **list()** de l'objet **query** est la plus générale pour exécuter une requête et constituer le résultat.
- Si vous êtes sûr que ce dernier ne contient qu'un seul objet, vous pouvez utiliser **uniqueResult()** :

```
session.createQuery("from Film where titre='Vertigo'")
.uniqueResult();
```

- Attention, une exception est levée si plus d'une ligne est trouvée.
- Cette méthode ne devrait être appliquée que pour des recherches portant sur des attributs déclarés **uniques** dans le schéma de la base de données (dont la clé primaire).