



# Formation Architecture Logicielle

## *Exercices*

Formateurs :

Frantz Degrigny, [frantz.degrigny@keyconsulting.fr](mailto:frantz.degrigny@keyconsulting.fr)

Méric Garcia, [meric.garcia@keyconsulting.fr](mailto:meric.garcia@keyconsulting.fr)

Téléphone : 02.30.96.02.75

<http://www.keyconsulting.fr>

## Sommaire

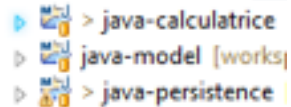
### 2. Découplez ! 3

1. EXERCICE 1 : PERSISTANCE DANS UN FICHIER
2. EXERCICE 2 : CHANGER LE STOCKAGE : POSTGRESQL
3. EXERCICE 3 : MODIFICATION D'UNE PROPRIÉTÉ
4. EXERCICE 4 : GESTION AUTOMATIQUE DE LA PERSISTANCE D'UN OBJET SIMPLE
5. EXERCICE 5 : UTILISATION D'UN ORM - HIBERNATE
6. EXERCICE 6 : UTILISATION D'UN ORM - CHANGEMENT DE LA BASE DE DONNÉES
7. EXERCICE 7 : UTILISATION D'UN ORM - PERFORMANCE
8. EXERCICE 8 : UNE AUTRE FAÇON D'ACCÉDER AU DONNÉES : JCR

## 2. Découplez !

### 1. Exercice 1 : Persistance dans un fichier

Nous avons un programme composé de trois modules :



```
> java-calculatrice  
▶ java-model [works]  
▶ java-persistence |
```

java-model : notre modèle objet

java-persistence : module gérant la persistance

java-calculatrice : vue et contrôleur du programme (couches supérieures)

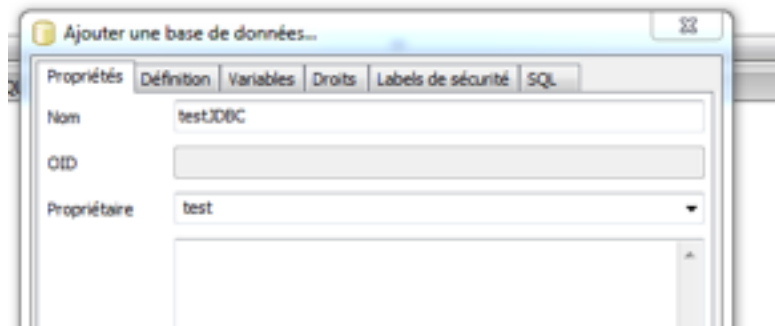
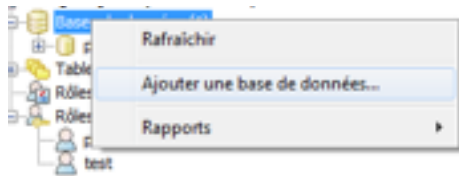
Récupérer la version de départ : git checkout EX1

Compiler le projet grâce à maven : mvn clean install

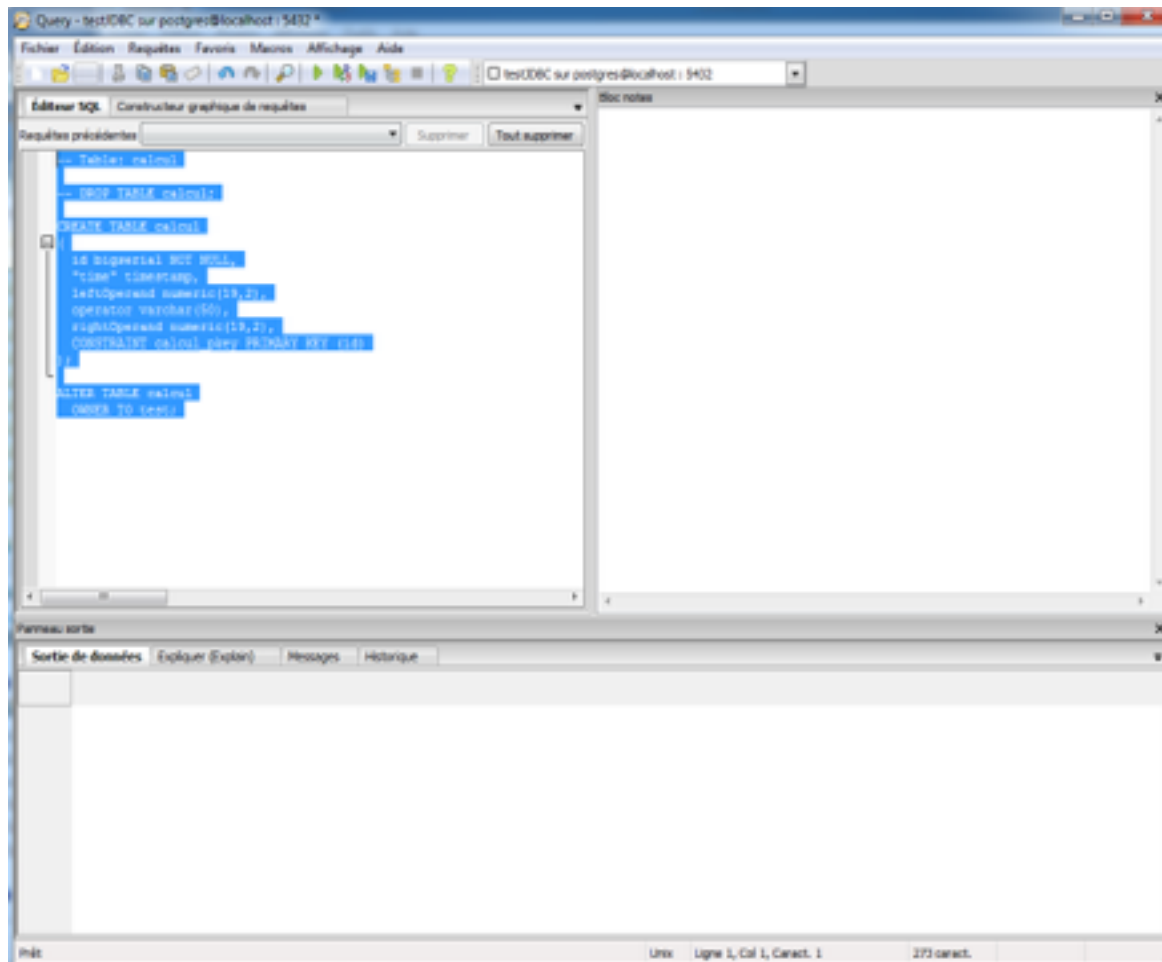
Lancer le projet dans Eclipse (le Main se trouve dans le projet java-calculatrice).

Implémenter une nouvelle fonctionnalité : les résultats doivent être persisté dans un fichier et rechargé au démarrage de l'application.





Lancer le script testJDBC.sql disponible dans le projet afin de créer la table.



Se placer au point de départ : git checkout -f EX2

Ajouter cette dependency dans le pom.xml du projet persistence.

```
<dependency>
```

```
    <groupId>postgresql</groupId>
```

```
    <artifactId>postgresql</artifactId>
```

```
    <version>9.1-901-1.jdbc4</version>
```

```
</dependency>
```

Implémenter une nouvelle fonctionnalité : les résultats doivent être persisté dans la nouvelle base de données. Utiliser le JDBCPersistenceService.

### 3. Exercice 3 : Modification d'une propriété

Se placer au point de départ : git checkout -f EX3

Ajouter le champ texte commentary au model.

Ajouter une entrée dans l'application javaFx afin de pouvoir le renseigner.

Modifier getCalculAsString pour retourner cette valeur.

Modifier la couche de persistence afin de gérer le nouvel objet calcul.

Modifier le schéma en base afin de pouvoir l'utiliser.

#### 4. Exercice 4 : Gestion automatique de la persistance d'un objet simple

Nous allons réutiliser le modèle précédent.

Se placer au point de départ : git checkout -f EX4

Modifier le MyORMPersistenceService afin d'obtenir les même services que pour JDBCPersistenceService.

##### Indications :

*Utiliser les méthodes utilitaires présentes dans la classe DBUtils.*

*Ces méthodes sont basées sur la réflexivité de java et la librairie commons-beanutils d'apache qui facilite son utilisation. L'annotation @Attribute a également été créée dans le modèle afin de mettre un flag sur les attributs de classes qui doivent être persistés.*

Une fois le service fonctionnel, ajouter une propriété supplémentaire (nouveau champ text du modèle et de l'application javaFx) : author. Modifier le code de l'application javaFX pour pouvoir l'ajouter. Modifier le modèle pour que getCalculAsString retourne l'author et le commentaire.

Relancer l'application et vérifier le bon fonctionnement de la persistance.

## 5. Exercice 5 : Utilisation d'un ORM - Hibernate

git checkout -f EX5

Nous allons réutiliser le modèle précédent.

Créer une nouvelle base de données (test) avec le même user que précédemment.

Modifier le ORMPersistenceService afin d'obtenir les mêmes services que pour JDBCPersistenceService.

Examiner le fichier java-persistence/src/main/resources/META-INF/persistence.xml.

- les classes dont hibernate doit avoir connaissance sont renseignées
- la configuration de la connexion à la base de données y figure aussi
- Modifier la propriété hibernate.hbm2ddl.auto :

```
<property name="hibernate.hbm2ddl.auto" value="update" />
```

Ajouter une propriété au modèle et à l'application javafx (second\_commentary) et relancer.

## 6. Exercice 6 : Utilisation d'un ORM - Changement de la base de données

Modifions la configuration pour utiliser une nouvelle base de donnée : apache derby (base de données locale sous forme de fichier).

Une référence dans le pom.xml permet d'obtenir le driver :

```
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.11.1.1</version>
</dependency>
```

Nous allons devoir modifier les propriétés suivantes du fichier persistence.xml :

```
<property name="connection.driver_class" value="org.apache.derby.jdbc.EmbeddedDriver" />
<property name="javax.persistence.jdbc.url" value="jdbc:derby:simpleDb;create=true" />
<property name="hibernate.default_schema" value="app" />
<property name="hibernate.dialect" value="org.hibernate.dialect.DerbyDialect" />
```

Vérifier que la persistance fonctionne toujours.



## 7. Exercice 7 : Utilisation d'un ORM - Performance

Modifions la configuration pour pouvoir examiner les logs.

```
<property name="show_sql" value="true" />
```

Ajouter un fichier de configuration pour les logs :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="console" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{ABSOLUTE} [%t] %-5p %c{1} - %m%n"/>
    </layout>
  </appender>

  <!-- Limit the org.apache category -->
  <category name="org.hibernate.SQL">
    <priority value="debug"/>
  </category>

  <root>
    <priority value="INFO" />
    <appender-ref ref="console" />
  </root>
</log4j:configuration>
```

Observer la génération de code SQL.

Nous allons remplacer la requête de chargement des calculs afin de pouvoir gérer la façon de récupérer les entités liées.

```
@SuppressWarnings("unchecked")
@Override
public List<Calcul> load() {
    Session session = em.unwrap(Session.class);
    Criteria crit = session.createCriteria(Calcul.class);
    List<Calcul> calculs = crit.list();
    return calculs;
}
```

Relancer l'application.

Observer le code SQL généré.

Utiliser l'annotation @Fetch pour retrouver le comportement précédent.

## 8. Exercice 8 : Une autre façon d'accéder au données : JCR

Installation :

Installons un site utilisant JCR.

Suivre les instructions d'installation : <http://www.onehippo.org/trails/getting-started/hippo-essentials-getting-started.html>

Une fois le site lancé, se connecter au cms (admin - admin) : localhost:8080/cms

Exercice :

Suivre les indications suivantes pour créer deux documents :

- Cliquer sur « Accéder aux documents »
- Cliquer sur « myhippoproject »
- Cliquer sur la flèche à coté de myhippoproject et sélectionner « nouveau document »
- Créer deux documents de chaque type disponible (liste de valeurs et ressources bundle)

Se connecter à la console : localhost:8080/cms/console

Exporter un nœuds, modifier quelques propriétés et l'importer.

Se connecter au repository : localhost:8080/cms/repository

Ecrire et effectuer les requêtes suivantes :

XPATH : récupérer tous les documents de type « » sous le noeuds document.

SQL : récupérer tous les noeuds de type « » créé aujourd'hui.