

MODULE 6

Applications n-tiers / Exercices

Exercice 1 : Entité simple

Exercice 1 : entité simple

On veut définir un type d'entité (objet métier persistant) qui modélise une **catégorie** (dans le sens du projet Fil Rouge).

Une catégorie comporte une seule information : son nom.

On veut enregistrer les entités dans une base MySQL (la base "formation" par exemple).

On veut de plus que le provider ORM soit EclipseLink, et que chaque fois qu'on exécute le programme, la table contenant les données soit créée si nécessaire mais pas vidée.

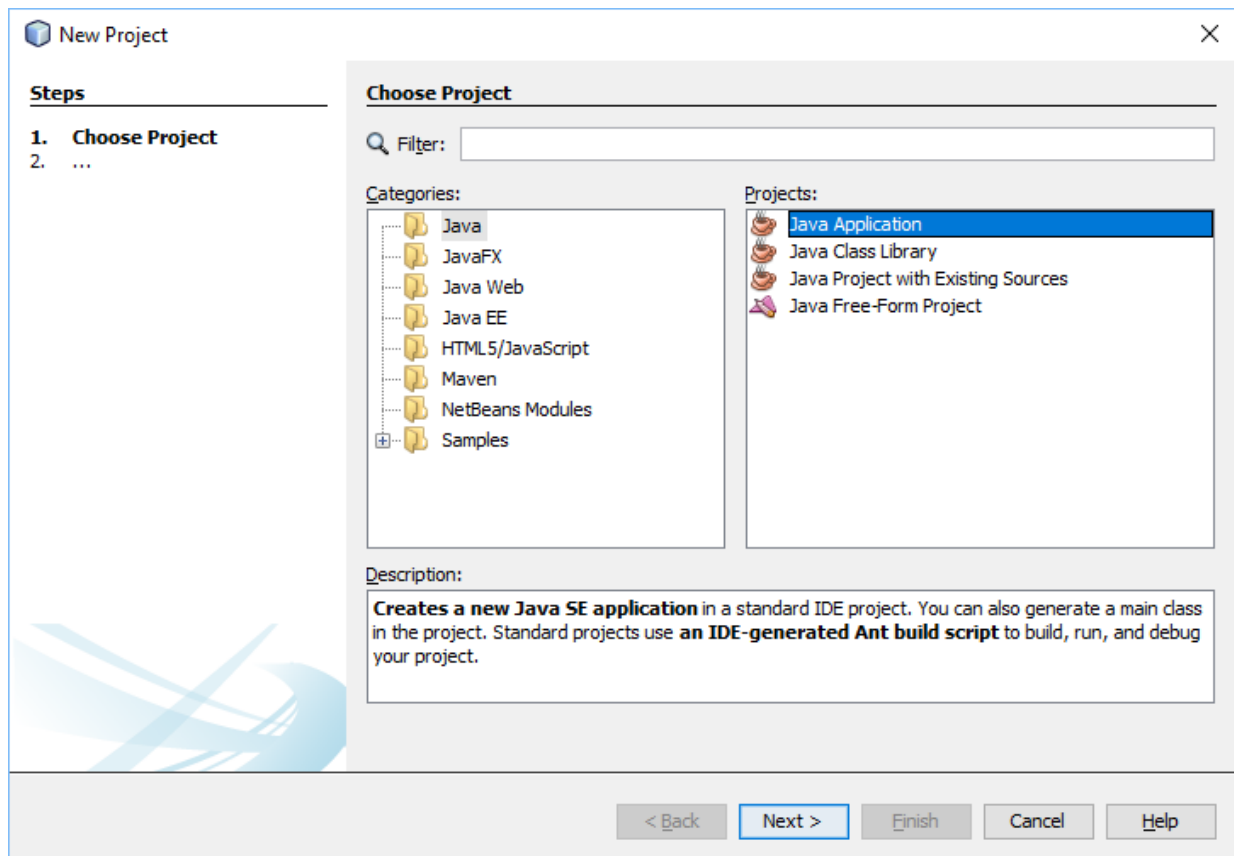
Exercice 1 : entité simple

On veut de plus que le code soit testé au moyen de JUnit. La méthode utilisée pour tester doit insérer 5 catégories différentes dans la base de données puis les récupérer et vérifier que le nombre d'entités dans la base est bien 5.

Le type de projet NetBeans est une simple application Java.

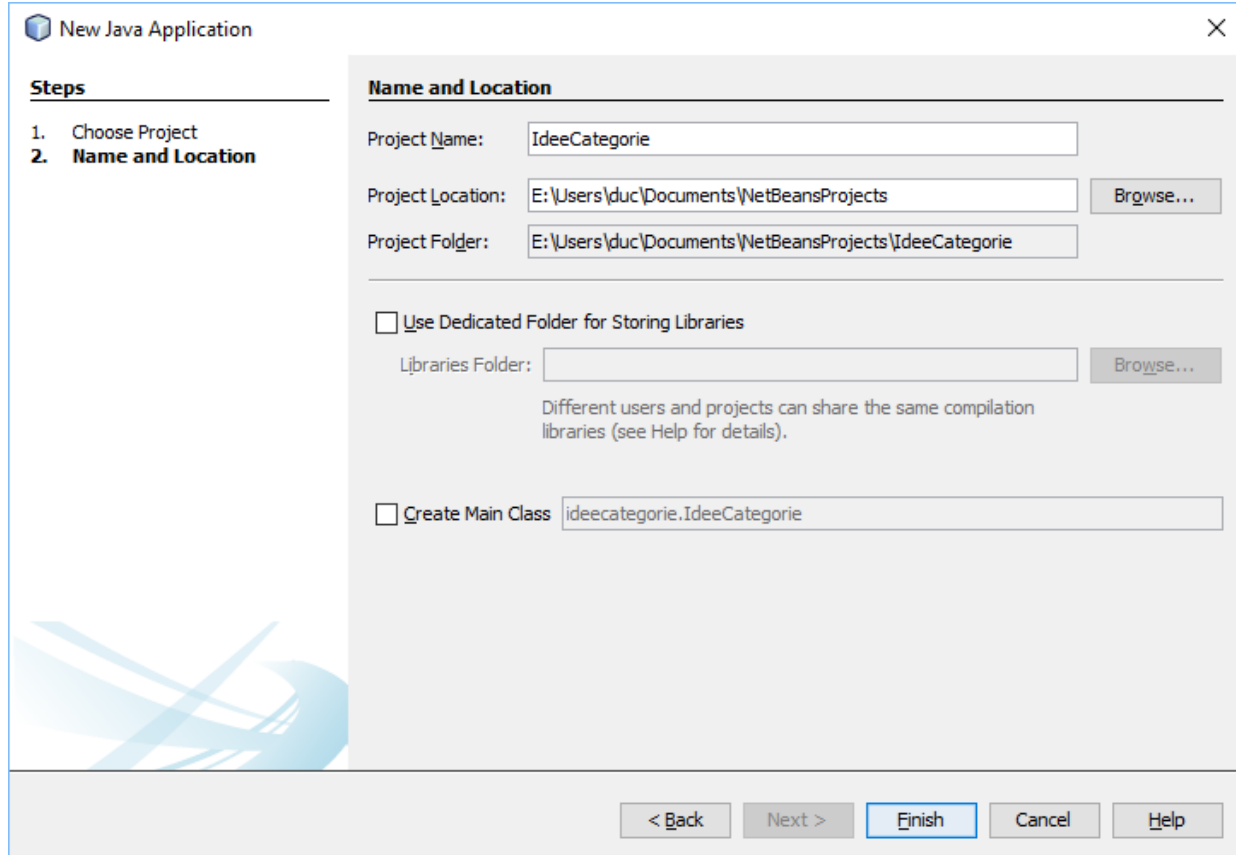
Exercice 1 : entité simple

Création du projet IdeeCategorie



Exercice 1 : entité simple

Création du projet IdeeCategorie (suite)



The screenshot shows the 'New Java Application' dialog box in NetBeans. The 'Steps' pane on the left indicates the current step is '2. Name and Location'. The 'Name and Location' tab is active, showing the following fields and options:

- Project Name:** IdeeCategorie
- Project Location:** E:\Users\duc\Documents\NetBeansProjects (with a 'Browse...' button)
- Project Folder:** E:\Users\duc\Documents\NetBeansProjects\IdeeCategorie
- ☐ **Use Dedicated Folder for Storing Libraries**
Libraries Folder: (with a 'Browse...' button)
Different users and projects can share the same compilation libraries (see Help for details).
- ☐ **Create Main Class** ideecategorie.IdeeCategorie

At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish' (highlighted), 'Cancel', and 'Help'.

Exercice 1 : entité simple

Nous allons utiliser MySQL comme SGBDR.

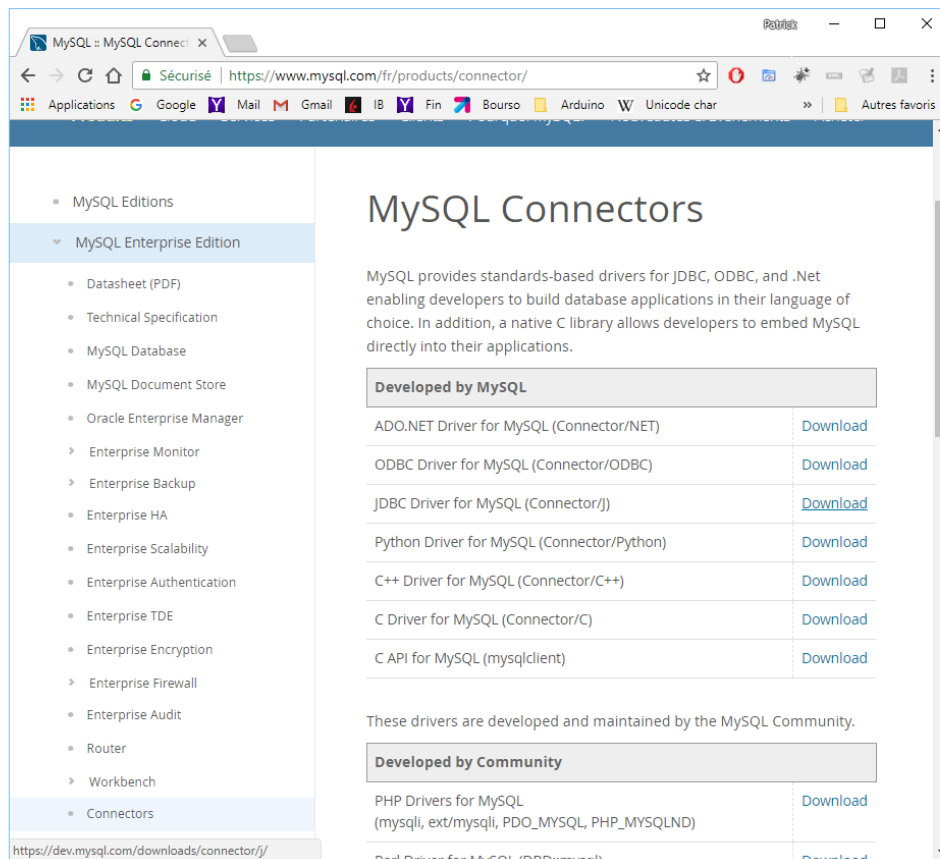
Il faut donc indiquer le driver JDBC de MySQL comme dépendance .

Ce driver est disponible sur le site de MySQL :

`https://www.mysql.com/fr/products/connector/`

Exercice 1 : entité simple

Récupération du driver JDBC de MySQL



The screenshot shows the MySQL Connectors page. The left sidebar lists various MySQL products, with 'MySQL Enterprise Edition' expanded. The main content area is titled 'MySQL Connectors' and describes the standard-based drivers for JDBC, ODBC, and .Net. It includes a table of drivers developed by MySQL, with links to download each driver. Below this, it mentions that other drivers are developed by the MySQL Community and provides a link to the community drivers page.

MySQL :: MySQL Connectors

https://www.mysql.com/fr/products/connector/

MySQL Connectors

MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to build database applications in their language of choice. In addition, a native C library allows developers to embed MySQL directly into their applications.

Developed by MySQL

ADO.NET Driver for MySQL (Connector/NET)	Download
ODBC Driver for MySQL (Connector/ODBC)	Download
JDBC Driver for MySQL (Connector/J)	Download
Python Driver for MySQL (Connector/Python)	Download
C++ Driver for MySQL (Connector/C++)	Download
C Driver for MySQL (Connector/C)	Download
C API for MySQL (mysqlclient)	Download

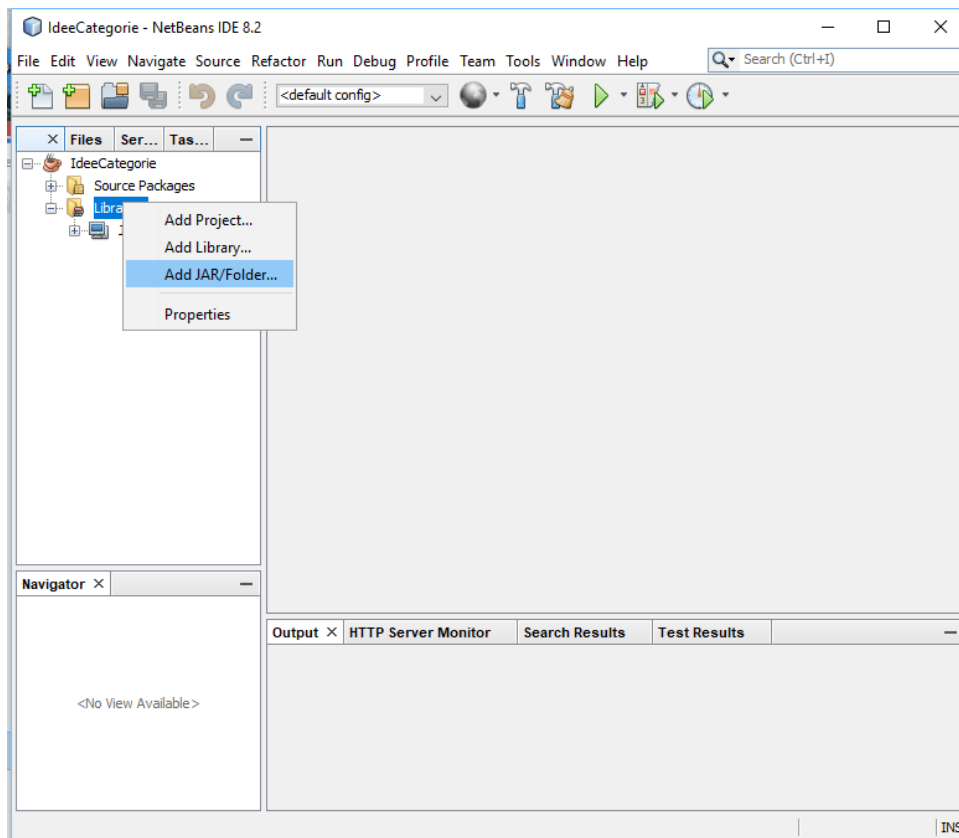
These drivers are developed and maintained by the MySQL Community.

Developed by Community

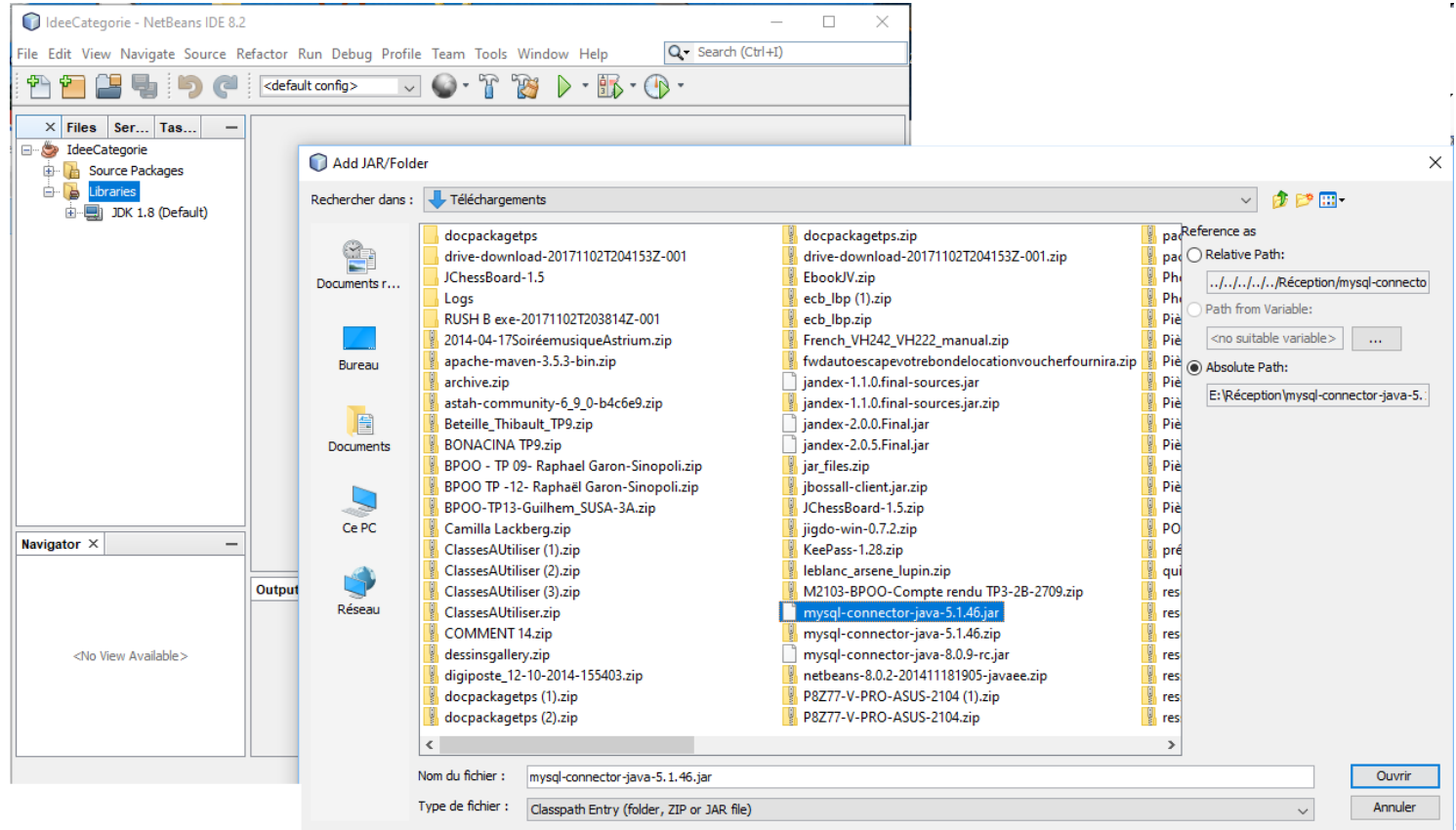
PHP Drivers for MySQL (mysql, ext/mysql, PDO_MYSQL, PHP_MYSQLND)	Download
Perl Driver for MySQL (DBD::mysql)	Download

Exercice 1 : entité simple

Ajout du driver
JDBC de
MySQL dans les
dépendances du
projet

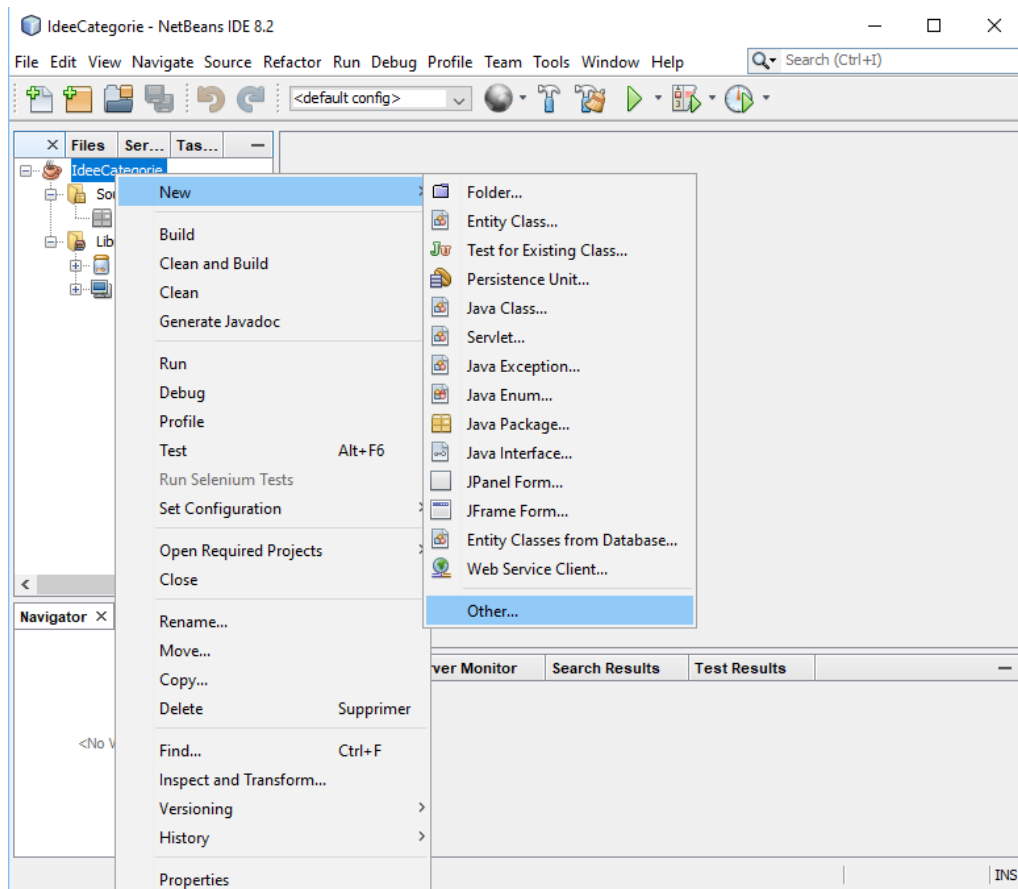


Exercice 1 : entité simple



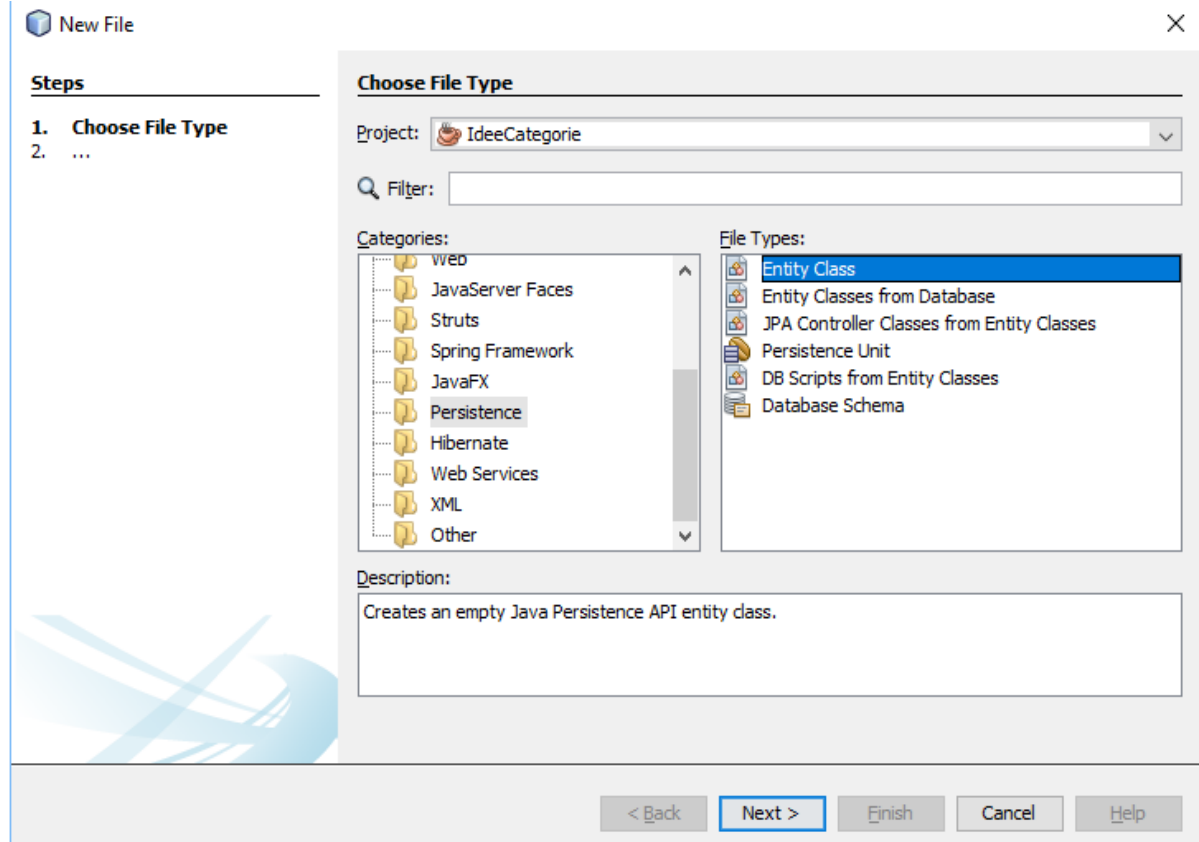
Exercice 1 : entité simple

Créer ensuite une entité (étape 1)



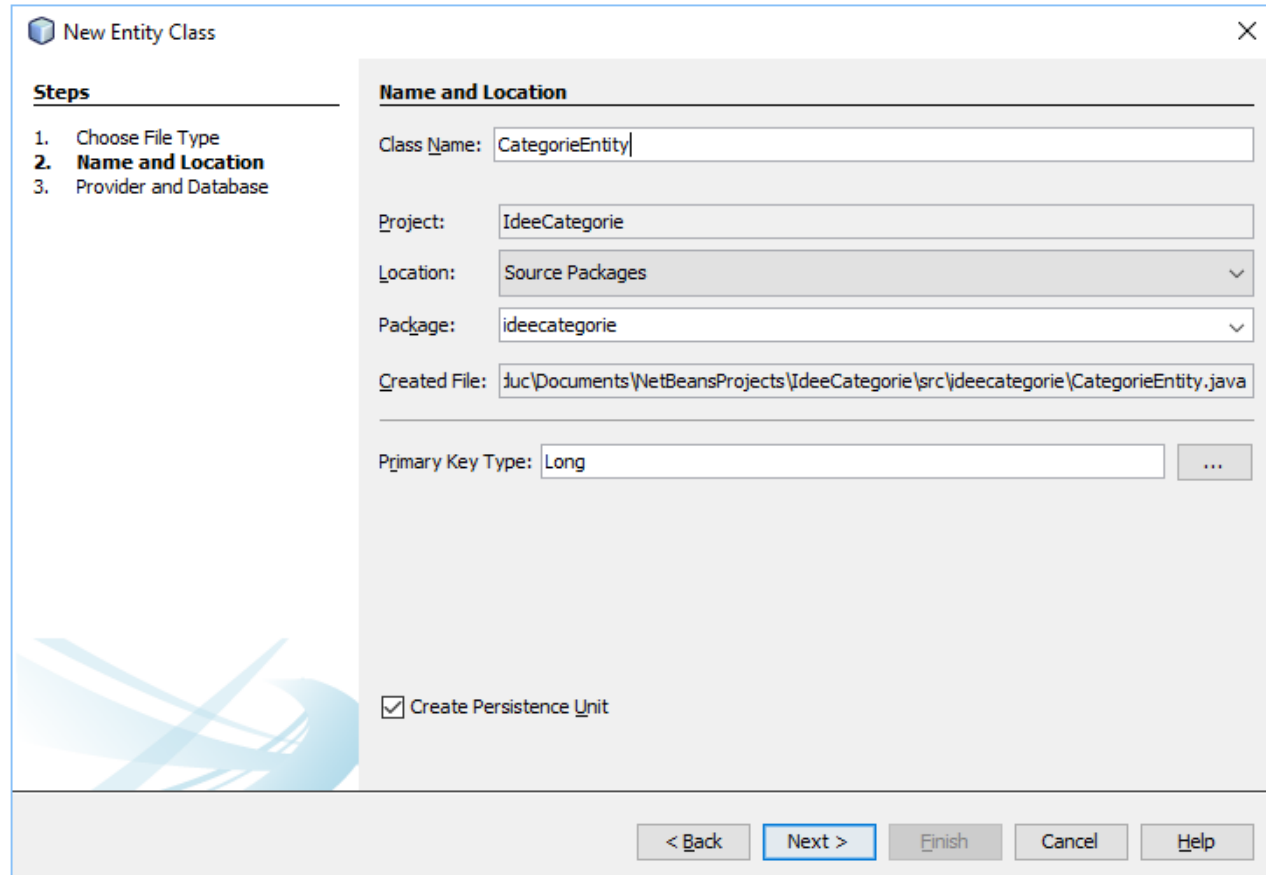
Exercice 1 : entité simple

Créer ensuite une entité (étape 2)



Exercice 1 : entité simple

Créer ensuite une entité (étape 3)



The screenshot shows the 'New Entity Class' dialog box in NetBeans IDE. The dialog is titled 'New Entity Class' and has a close button (X) in the top right corner. It is divided into two main sections: 'Steps' and 'Name and Location'.

Steps:

1. Choose File Type
2. **Name and Location**
3. Provider and Database

Name and Location:

Class Name:

Project:

Location:

Package:

Created File:

Primary Key Type: ...

☒ Create Persistence Unit

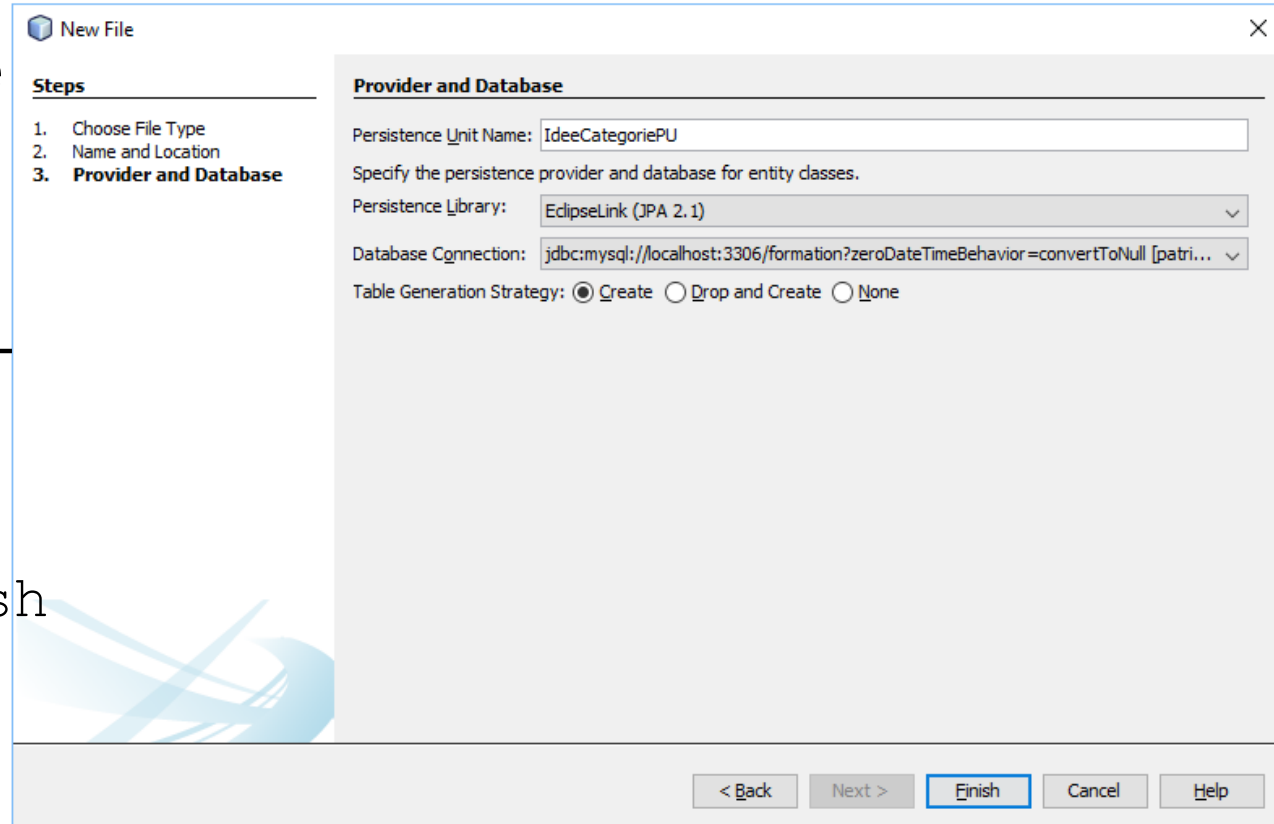
At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted with a blue border.

Exercice 1 : entité simple

Créer ensuite une entité (étape 4)

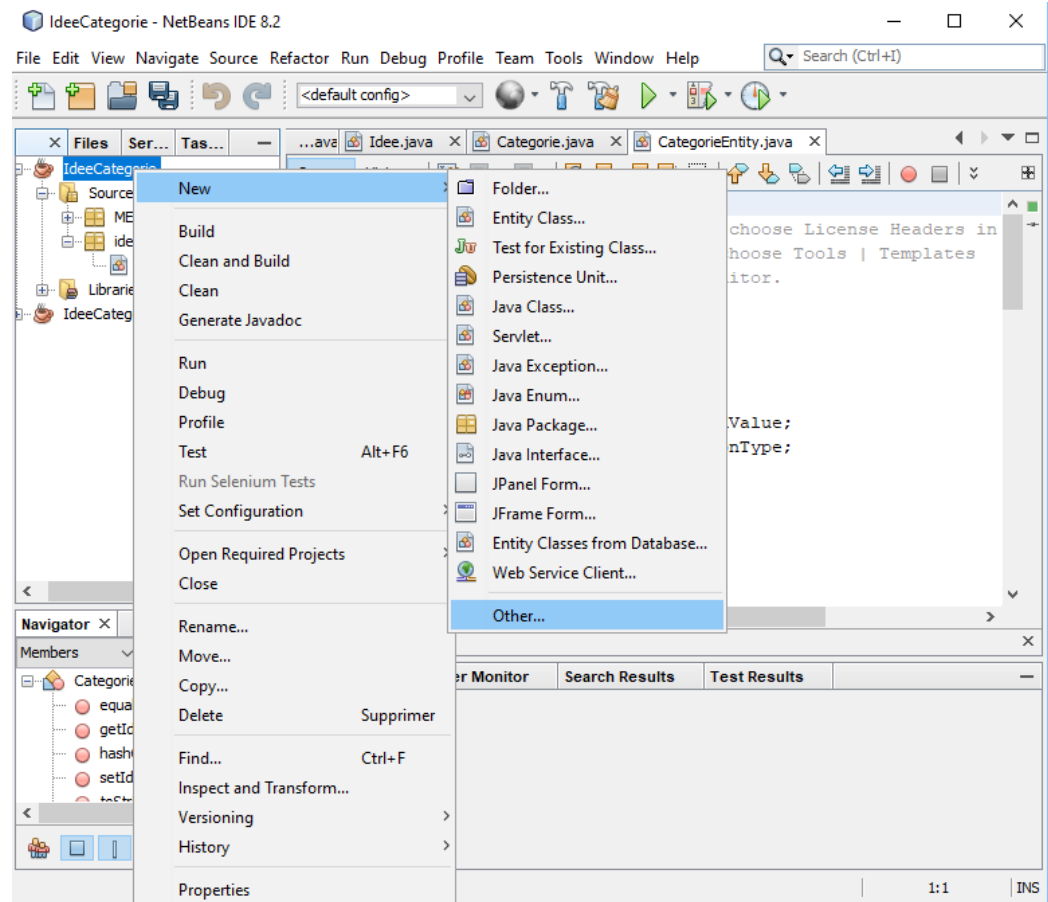
On va utiliser une connexion MySQL (datasource) existante

Cliquer sur Finish à la fin.



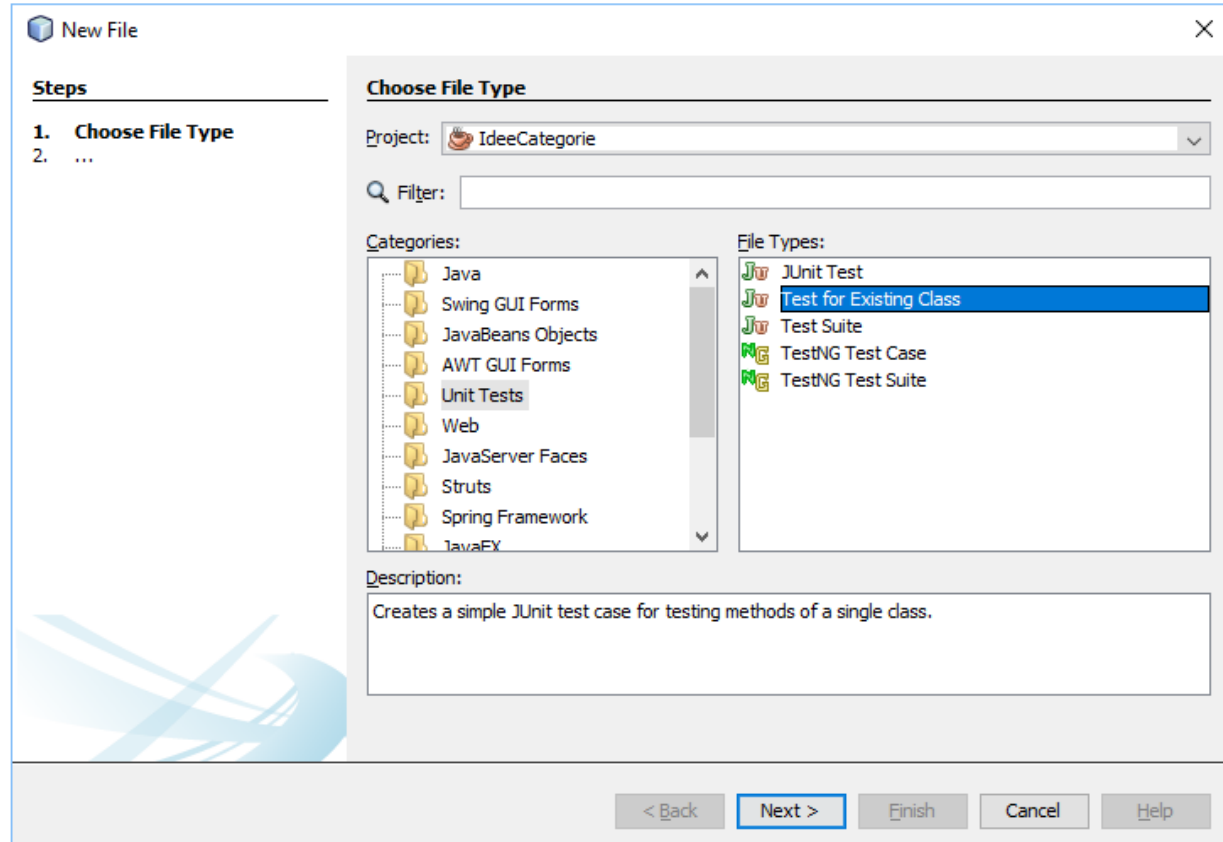
Exercice 1 : entité simple

Il faut ensuite créer une classe de test JUnit (étape 1).



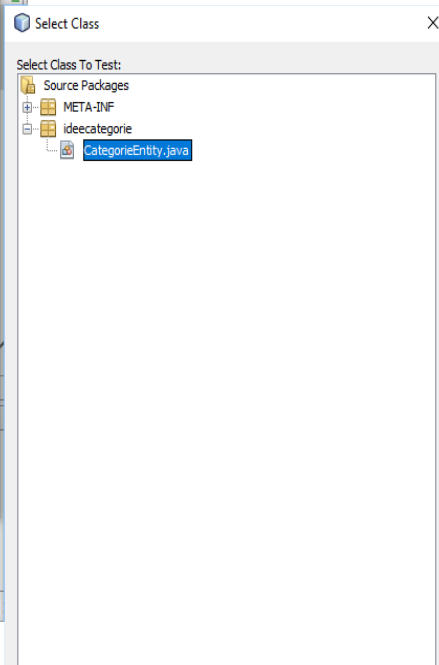
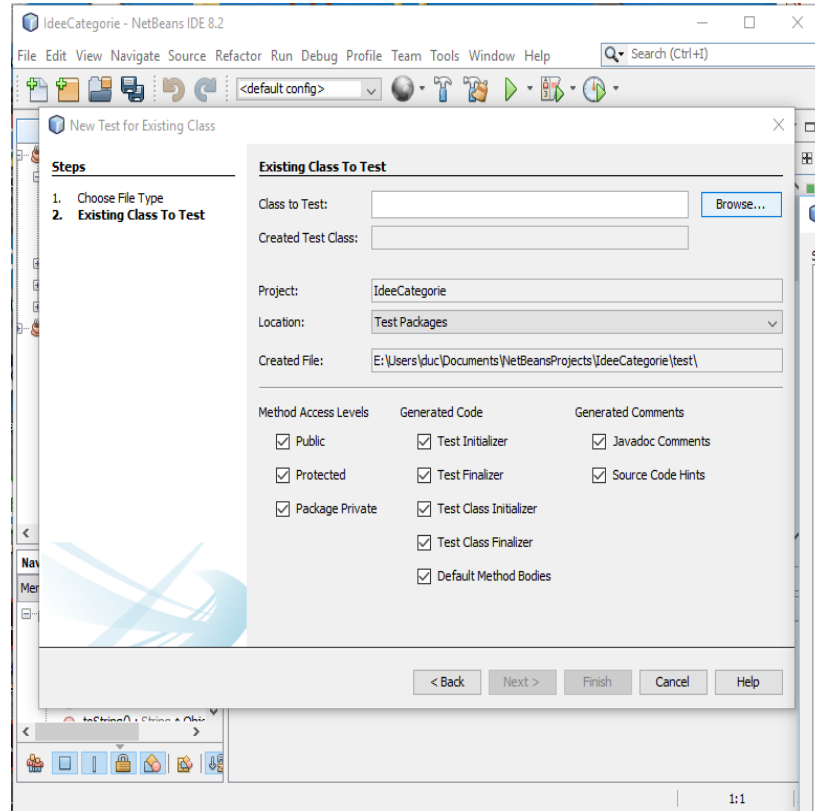
Exercice 1 : entité simple

Il faut ensuite créer
une classe de test
JUnit (étape 2).



Exercice 1 : entité simple

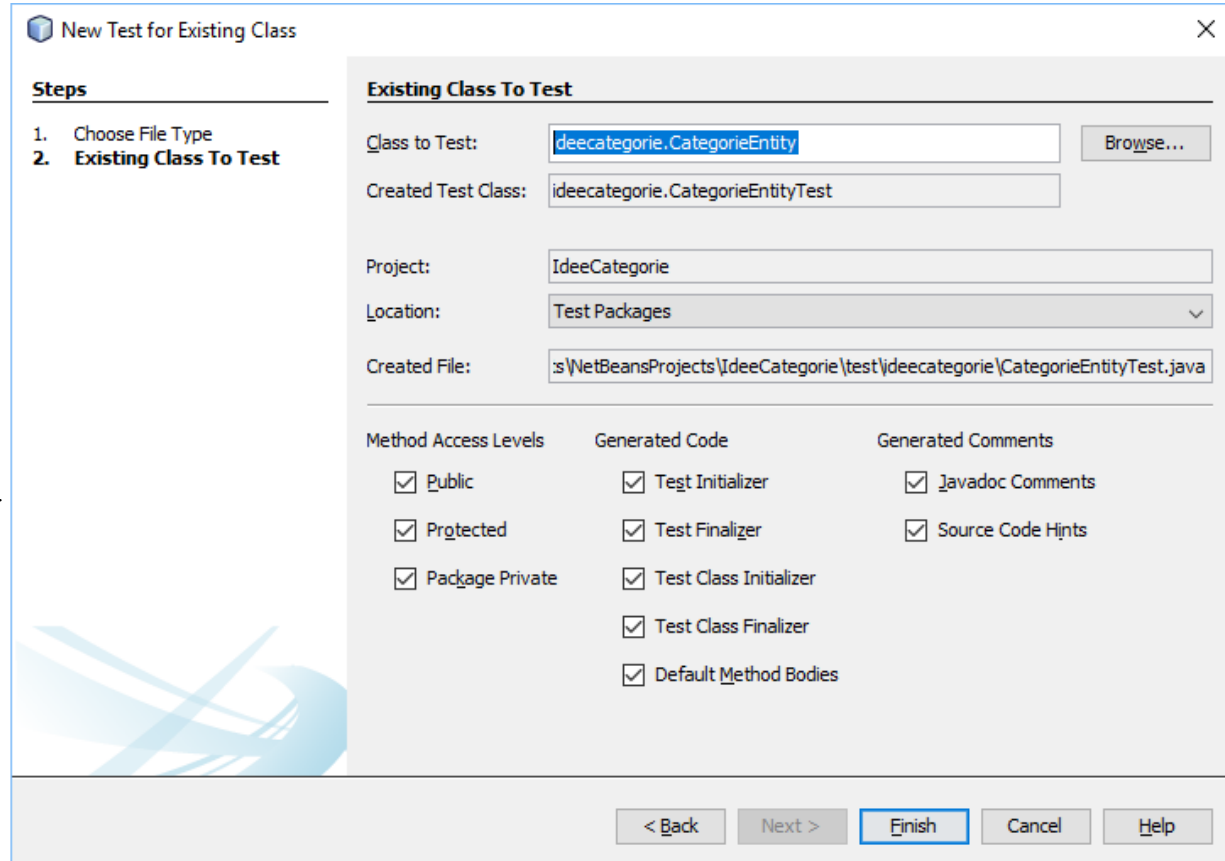
Il faut ensuite créer
une classe de test
JUnit (étape 3).



Exercice 1 : entité simple

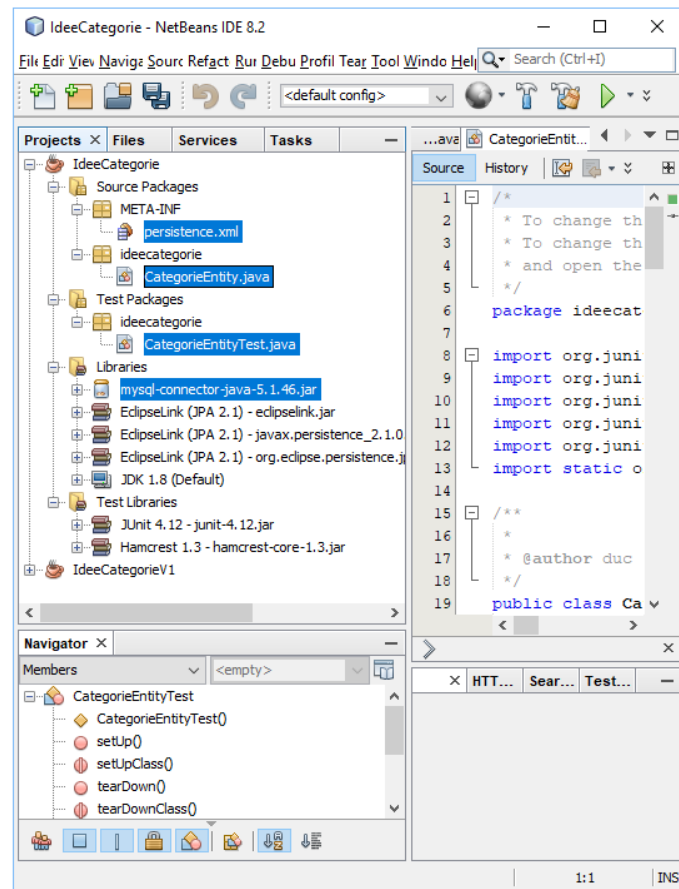
Il faut ensuite créer
une classe de test
JUnit (étape 4).

Cliquer sur `Finish`
à la fin.



Exercice 1 : entité simple

Organisation finale du projet.



Exercice 1 : entité simple

Ajouter l'annotation suivante au niveau de l'entité `Categorie` pour avoir la possibilité de lister toutes les instances de l'entité `CategoryEntity` dans la base :

```
@NamedQueries ({@NamedQuery (name =  
"CategorieEntity.findAll", query = "select e  
from CategorieEntity e") })
```

Exercice 1 : entité simple

A vous de développer la classe `CategorieEntity` et de la tester avec la classe de test JUnit `CategorieEntityTest` !

Exercice 2 : Entité simple avec champ à valeur unique

Exercice 2 : champ à valeur unique

Modifier l'entité `CategorieEntity` de manière à ce que l'unicité du nom de catégorie soit assurée.

Effacer la table `categorieentity` dans la base MySQL, puis relancer plusieurs fois le test JUnit sur l'entité.

Observer le résultat.

Exercice 3 : Relation 1-1 unidirectionnelle

Exercice 3 : relation 1-1 unidirectionnelle

On veut définir une entité `IdeeEntity` qui modélise schématiquement une idée telle que définie dans le projet Fil Rouge, et établir une relation unidirectionnelle entre une `Idee` et une `CategorieEntity`.

Une entité `IdeeEntity` contiendra comme champs un titre et une description, tous deux de type `String`, et établira une relation 1-1 unidirectionnelle vers une entité `CategorieEntity`.

Exercice 3 : relation 1-1 unidirectionnelle

On veut également pouvoir créer une `IdeeEntity` à la volée, lui associer une `CategorieEntity` non encore enregistrée dans la base de données, et que lorsque l'`IdeeEntity` est enregistrée, alors la `CategorieEntity` soit aussi enregistrée.

Par contre, si on supprime une `IdeeEntity`, on ne veut pas que sa `CategorieEntity` associée soit effacée.

Exercice 3 : relation 1-1 unidirectionnelle

Cette `IdeeEntity` sera testée au moyen d'un test JUnit qui consistera à créer plusieurs idées et catégories, en associant une catégorie chaque idée, et à enregistrer les entités dans le Persistence Context.

La liste des catégories disponibles sera récupérée dans le test depuis la base de données ; s'inspirer de la façon dont la liste des catégories dans la base de données est récupérée dans l'exercice 1.

Lorsque le test fonctionnera et que la table des idées sera remplie, essayer de supprimer la table des catégories.

Exercice 4 : Relation 1-n bidirectionnelle

Exercice 4 : relation 1-n bidirectionnelle

De manière plus réaliste que dans l'exercice précédent, on veut que plusieurs idées puissent appartenir à la même catégorie, ce qui veut dire établir une relation 1-n entre une `CategorieEntity` et une `IdeeEntity`.

De plus, on souhaite pouvoir parcourir la relation dans les deux sens : la relation sera de plus bidirectionnelle.

Exercice 4 : relation 1-n bidirectionnelle

Au travail !

Exercice 5 : Relation n-p bidirectionnelle

Exercice 5 : relation n-p bidirectionnelle

Implémenter une bibliothèque permettant de relier des entités « livre » et « auteur » dans le cadre d'une relation n-p bidirectionnelle.

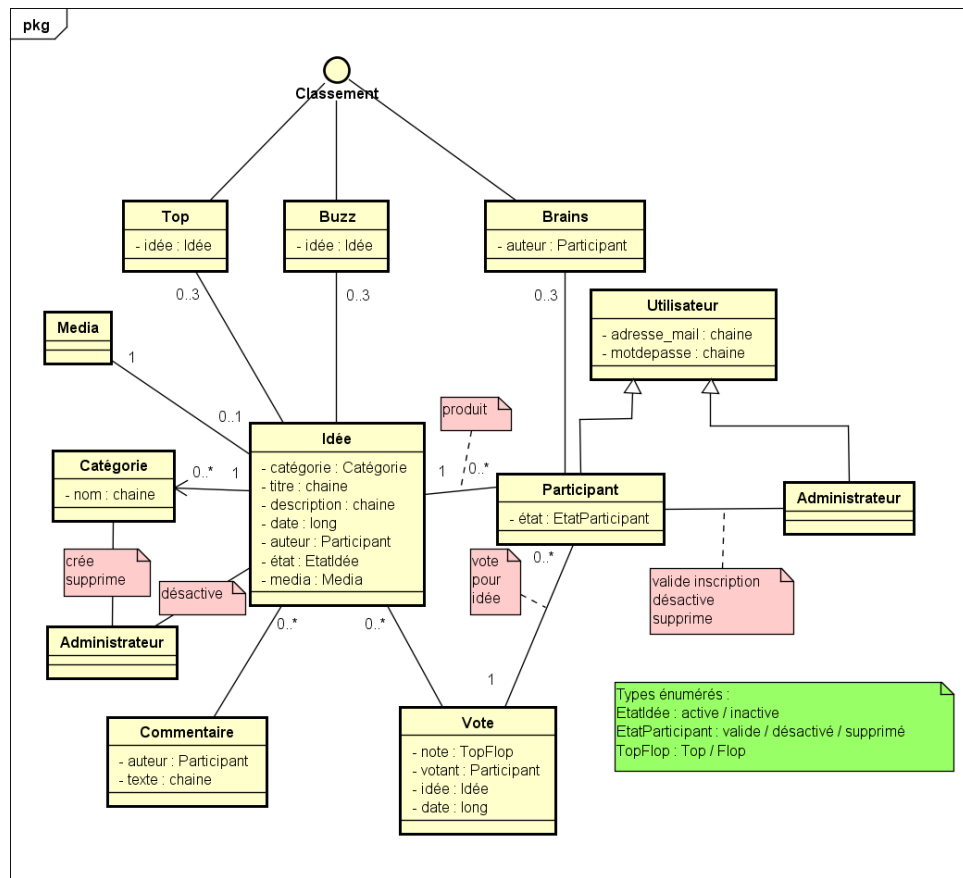
Exercice 6 : Retour au projet Fil Rouge

Exercice 6 : retour au projet Fil Rouge

Dans le cadre du projet FilRouge, une entité permet de relier deux autres entités dans le cadre d'une relation n-p.

Laquelle ?

Exercice 6 : retour au projet Fil Rouge



Exercice 6 : retour au projet Fil Rouge

Il s'agit du vote :

- un participant peut voter pour plusieurs idées
- plusieurs participants peuvent voter pour la même idée.

Exercice 7 : Un simple EJB Session

Exercice 7 : un simple EJB Session

On veut implémenter une calculatrice basique. Cette calculatrice supporte les 4 opérations de base sur des nombres de type `double` :

- addition
- soustraction
- multiplication
- division.

Quel type d'EJB Session nous faut-il utiliser ?

Exercice 7 : un simple EJB Session

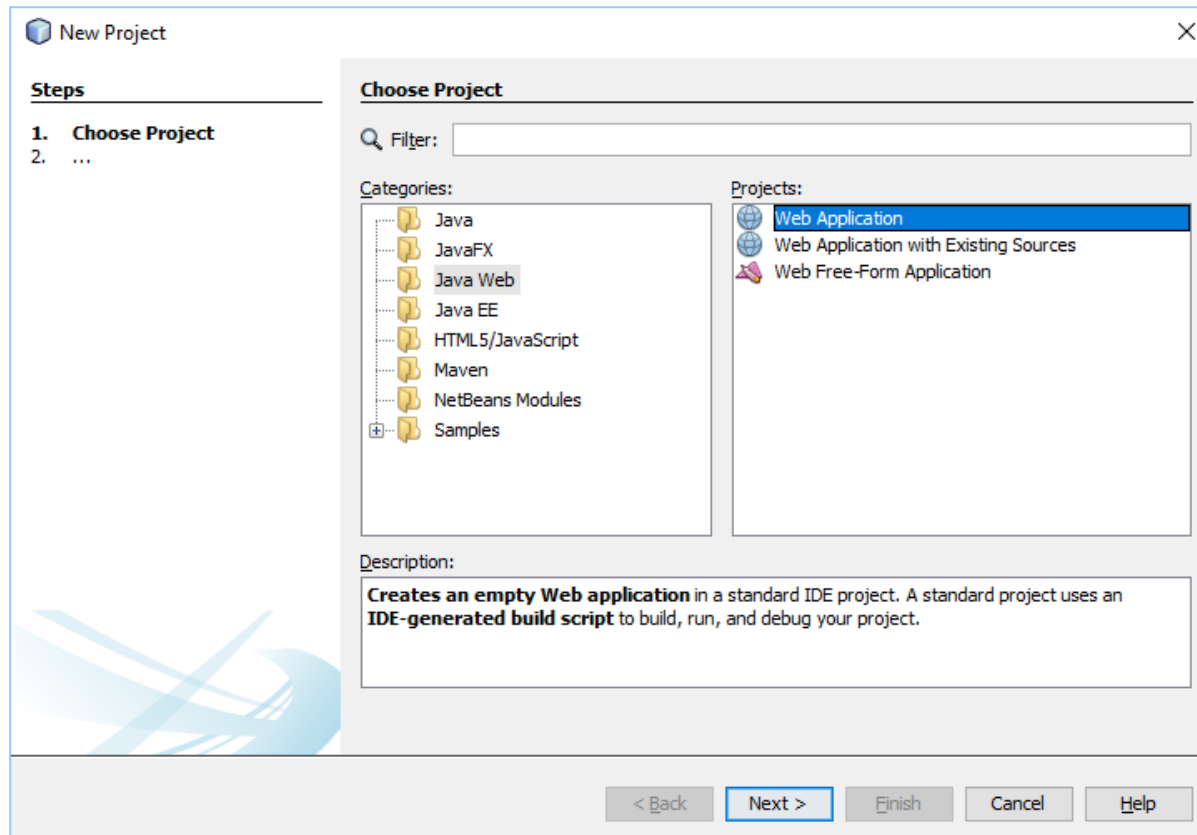
Vu que la calculatrice n'a pas besoin d'enregistrer des informations rémanentes entre deux appels, on peut utiliser un simple EJB Session stateless.

La calculatrice sera testée au moyen de JUnit.

On crée un nouveau projet dans NetBeans, de type WebApplication.

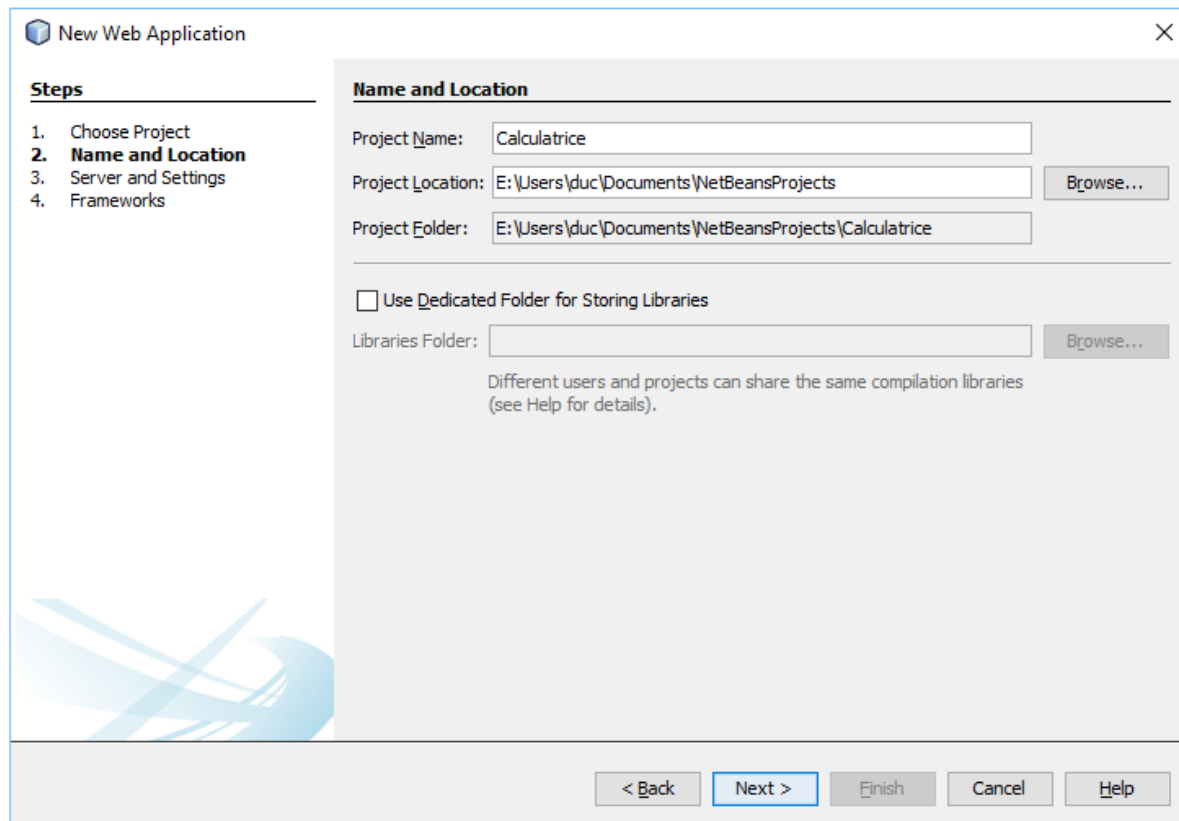
Exercice 7 : un simple EJB Session

Création d'un
projet de type
Web Application
(étape 1)



Exercice 7 : un simple EJB Session

Création d'un
projet de type
Web Application
(étape 2)



The screenshot shows the 'New Web Application' dialog box in NetBeans. The 'Steps' panel on the left lists four steps: 1. Choose Project, 2. **Name and Location**, 3. Server and Settings, and 4. Frameworks. The 'Name and Location' panel on the right contains the following fields and options:

- Project Name:** Calculatrice
- Project Location:** E:\Users\duc\Documents\NetBeansProjects (with a 'Browse...' button)
- Project Folder:** E:\Users\duc\Documents\NetBeansProjects\Calculatrice
- ☐ Use Dedicated Folder for Storing Libraries
- Libraries Folder:** (with a 'Browse...' button)

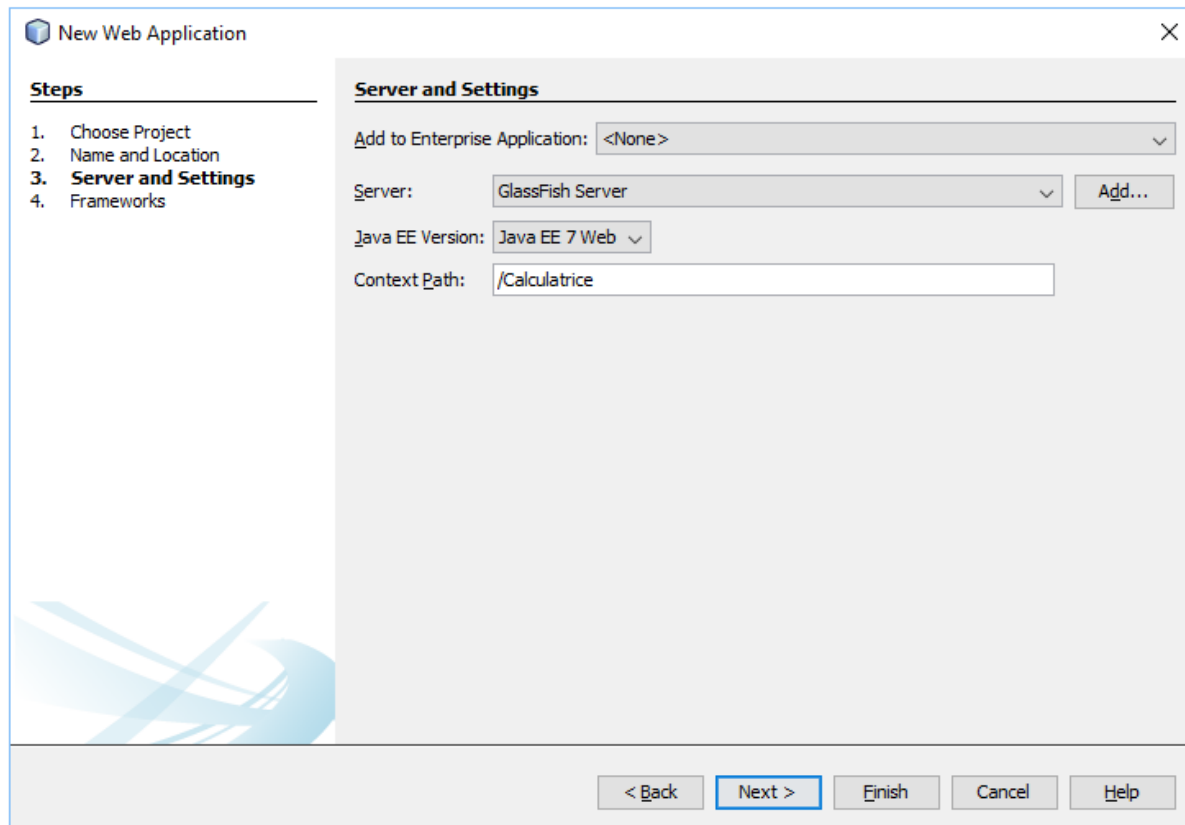
Below the 'Libraries Folder' field, there is a note: "Different users and projects can share the same compilation libraries (see Help for details)."

At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted with a blue border.

Exercice 7 : un simple EJB Session

Création d'un projet de type Web Application (étape 3)

NB. On utilise **GlassFish** comme serveur d'application pour une raison expliquée plus loin.



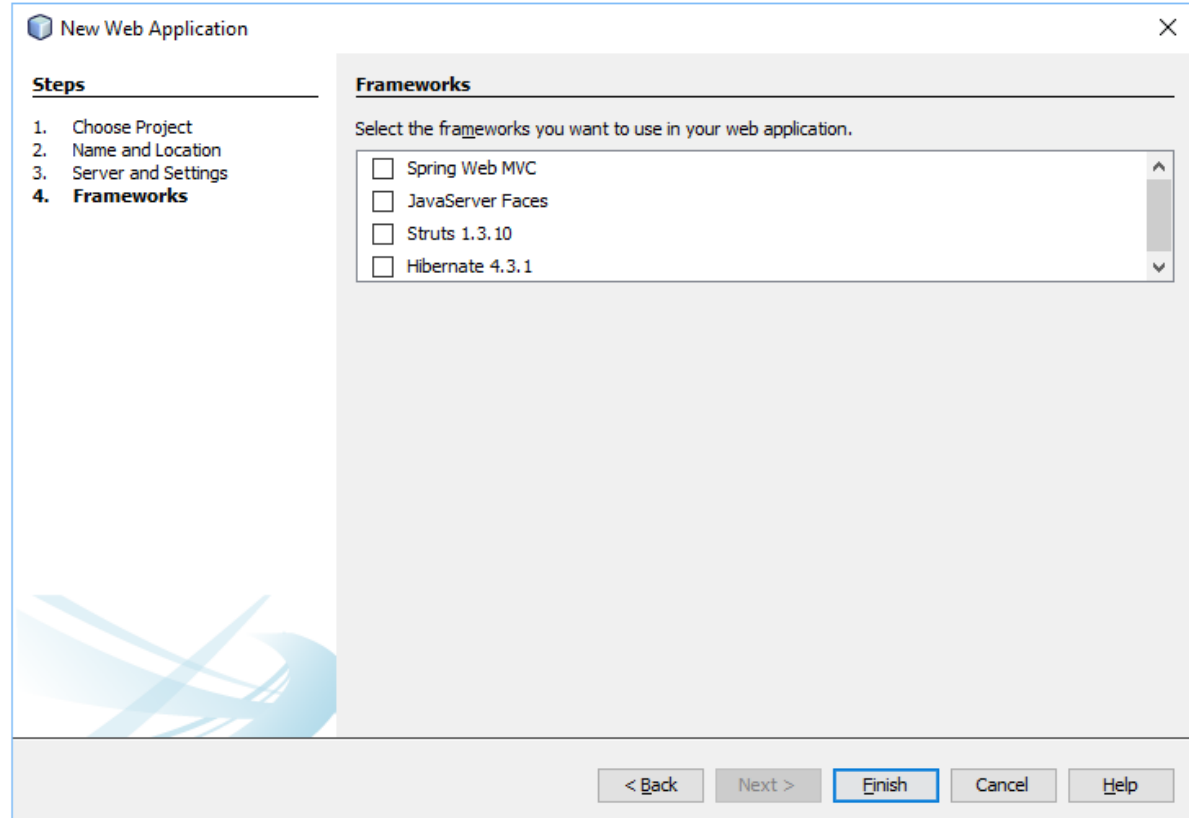
The screenshot shows the 'New Web Application' wizard in an IDE, specifically the 'Server and Settings' step. The wizard is titled 'New Web Application' and has a close button (X) in the top right corner. On the left, a 'Steps' panel lists four steps: 1. Choose Project, 2. Name and Location, 3. **Server and Settings** (the current step), and 4. Frameworks. The main area is titled 'Server and Settings' and contains the following fields and controls:

- 'Add to Enterprise Application:' with a dropdown menu showing '<None>'.
- 'Server:' with a dropdown menu showing 'GlassFish Server' and an 'Add...' button to the right.
- 'Java EE Version:' with a dropdown menu showing 'Java EE 7 Web'.
- 'Context Path:' with a text input field containing '/Calculatrice'.

At the bottom of the wizard, there are five buttons: '< Back', 'Next >' (which is highlighted with a blue border), 'Finish', 'Cancel', and 'Help'.

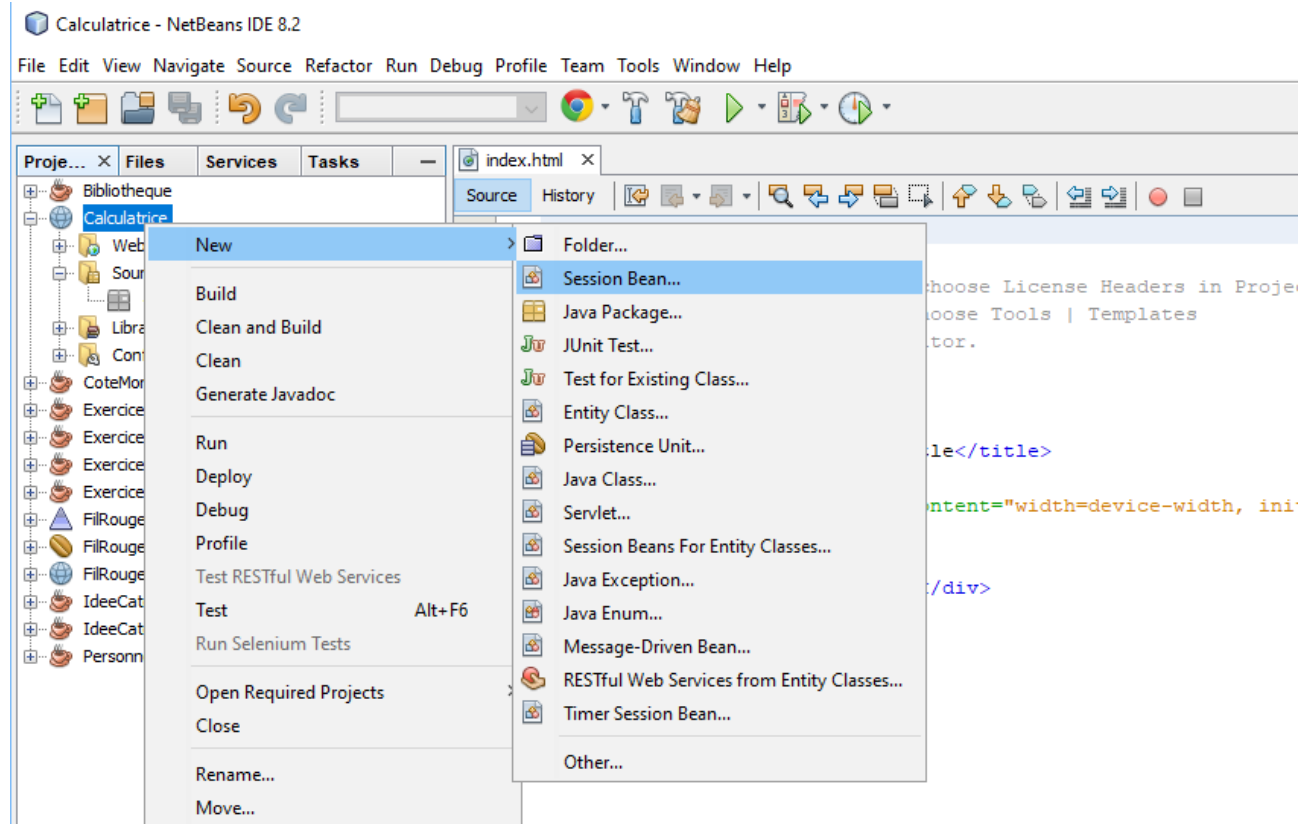
Exercice 7 : un simple EJB Session

Création d'un
projet de type
Web Application
(étape 4)



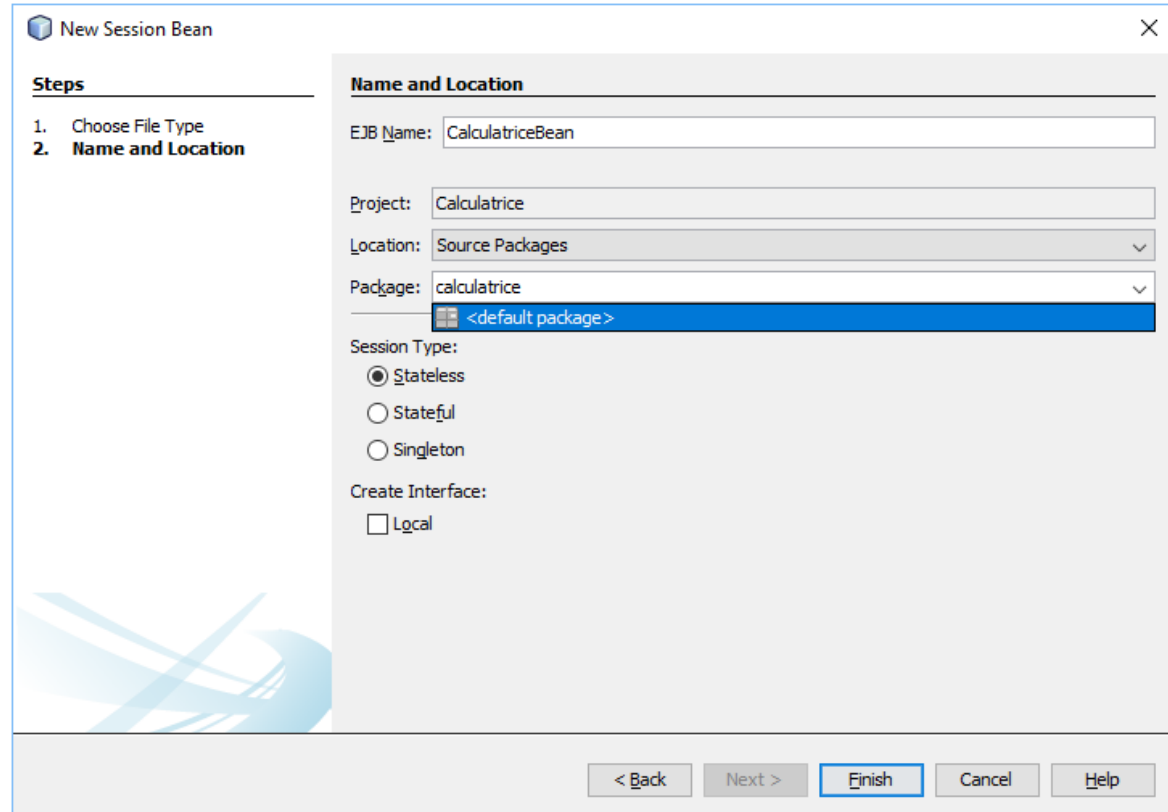
Exercice 7 : un simple EJB Session

Création d'un
EJB Session
(étape 1)



Exercice 7 : un simple EJB Session

Création d'un
EJB Session
(étape 2)



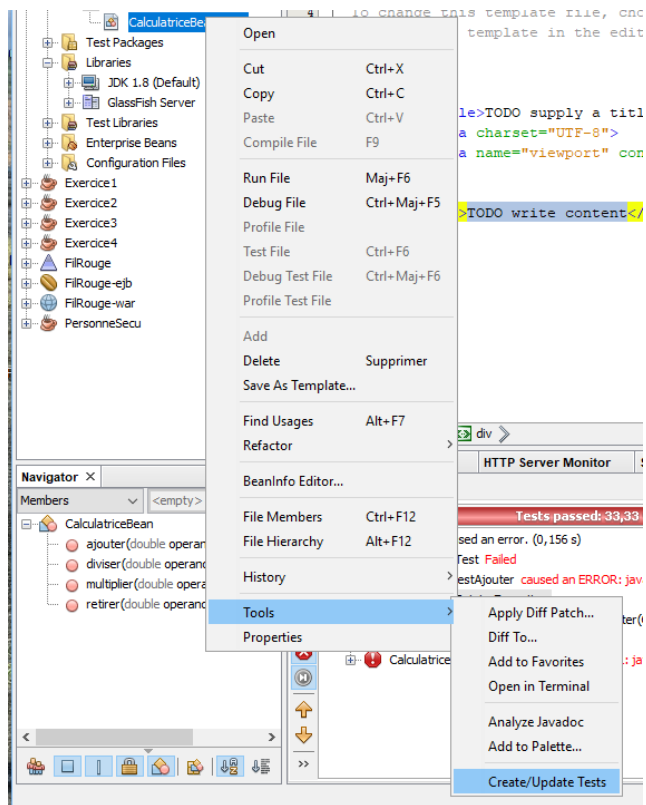
The screenshot shows the 'New Session Bean' dialog box with the following details:

- Steps:**
 1. Choose File Type
 2. **Name and Location**
- Name and Location:**
 - EJB Name:
 - Project:
 - Location:
 - Package:
 - Session Type:
 - ☒ Stateless
 - ☐ Stateful
 - ☐ Singleton
 - Create Interface:
 - ☐ Local

At the bottom, there are buttons: < Back, Next >, **Finish**, Cancel, and Help.

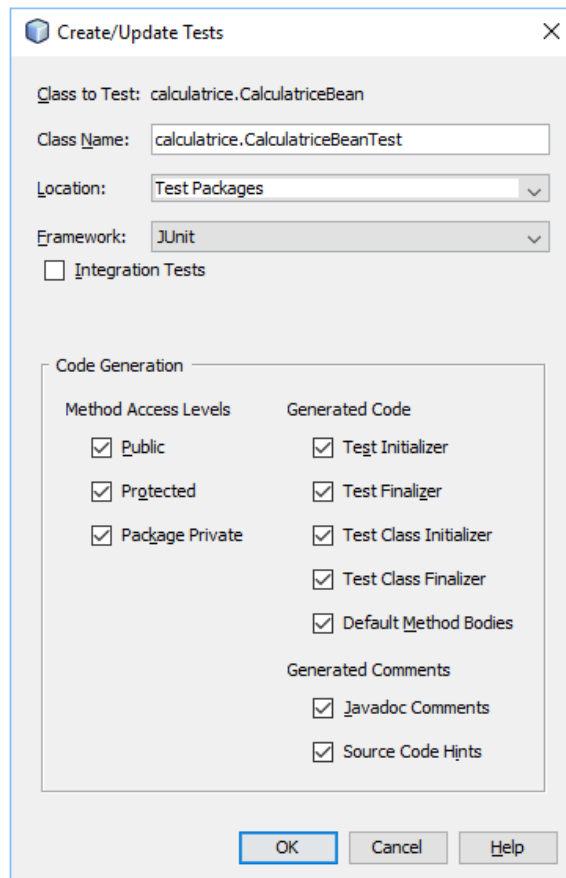
Exercice 7 : un simple EJB Session

Création d'un test
JUnit de l'EJB créé
(étape 1)



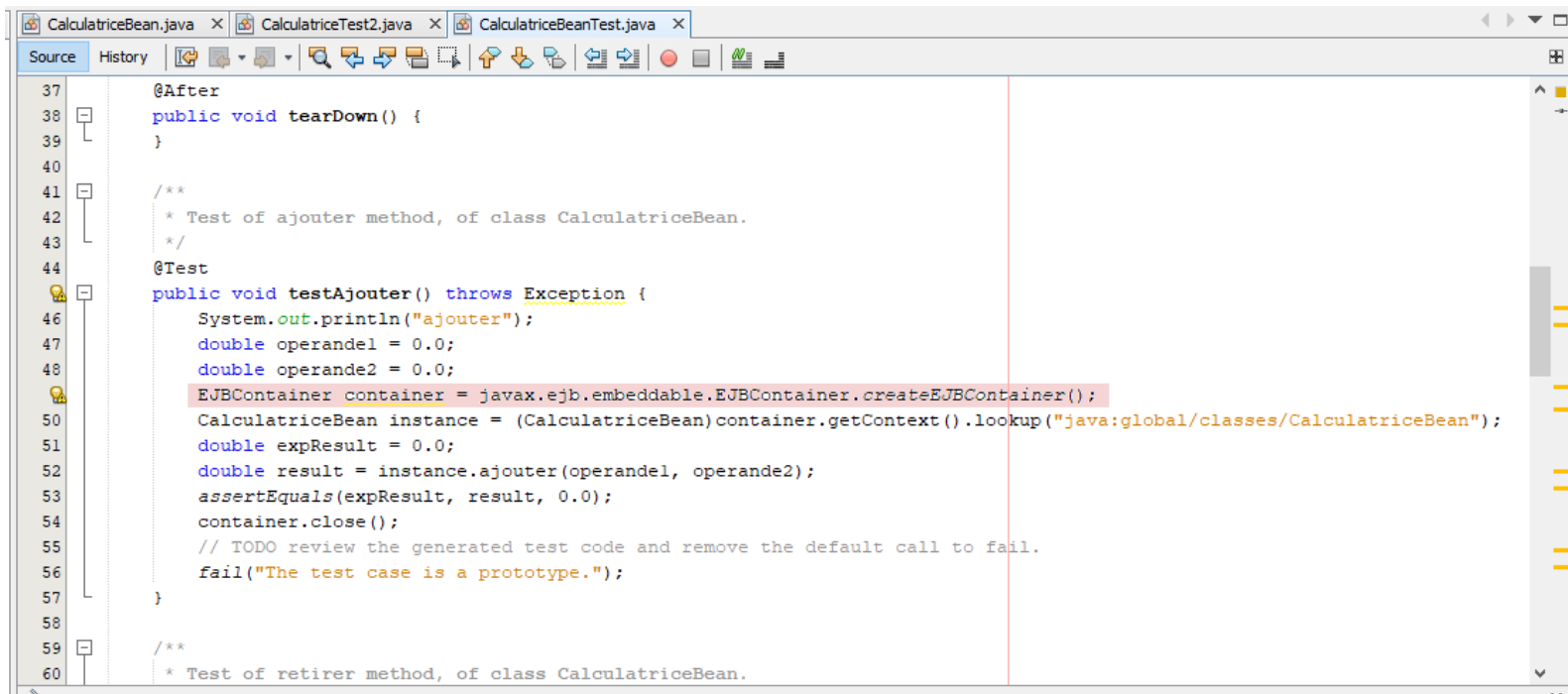
Exercice 7 : un simple EJB Session

Création d'un test
JUnit de l'EJB créé
(étape 2)



Exercice 7 : un simple EJB Session

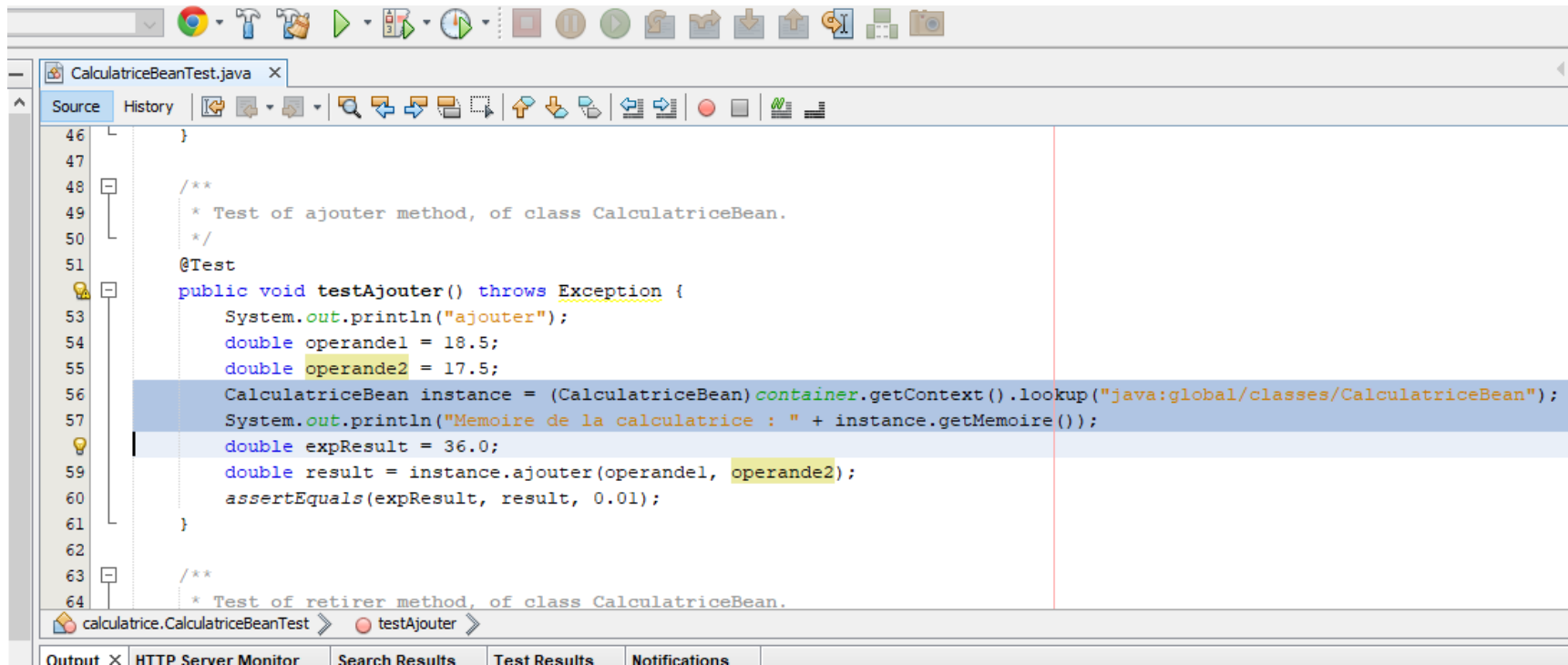
Ajout par NetBeans d'un container d'EJB embarquable – WildFly ne fournit pas de serveur embarquable, à la différence de GlassFish.



```
37  @After
38  public void tearDown() {
39  }
40
41  /**
42   * Test of ajouter method, of class CalculatriceBean.
43   */
44  @Test
45  public void testAjouter() throws Exception {
46      System.out.println("ajouter");
47      double operande1 = 0.0;
48      double operande2 = 0.0;
49      EJBContainer container = javax.ejb.embeddable.EJBContainer.createEJBContainer();
50      CalculatriceBean instance = (CalculatriceBean) container.getContext().lookup("java:global/classes/CalculatriceBean");
51      double expResult = 0.0;
52      double result = instance.ajouter(operande1, operande2);
53      assertEquals(expResult, result, 0.0);
54      container.close();
55      // TODO review the generated test code and remove the default call to fail.
56      fail("The test case is a prototype.");
57  }
58
59  /**
60   * Test of retirer method, of class CalculatriceBean.
```


Exercice 7 : un simple EJB Session

Récupération d'une instance de l'EJB CalculatriceBean



```
46     }
47
48     /**
49     * Test of ajouter method, of class CalculatriceBean.
50     */
51     @Test
52     public void testAjouter() throws Exception {
53         System.out.println("ajouter");
54         double operandel = 18.5;
55         double operande2 = 17.5;
56         CalculatriceBean instance = (CalculatriceBean)container.getContext().lookup("java:global/classes/CalculatriceBean");
57         System.out.println("Memoire de la calculatrice : " + instance.getMemoire());
58         double expResult = 36.0;
59         double result = instance.ajouter(operandel, operande2);
60         assertEquals(expResult, result, 0.01);
61     }
62
63     /**
64     * Test of retirer method, of class CalculatriceBean.
```

Exercice 7 : un simple EJB Session

On teste chaque méthode de l'EJB :

- `ajouter()`
- `retirer()`
- `multiplier()`
- `diviser()`

selon le modèle proposé par NetBeans, en retirant juste dans chaque test l'appel à `fail()` :

```
fail("The test case is a prototype.");
```

Exercice 7 : un simple EJB Session

Cependant, dans le test `testAjouter()`, on indique volontairement un résultat attendu **incorrect**.

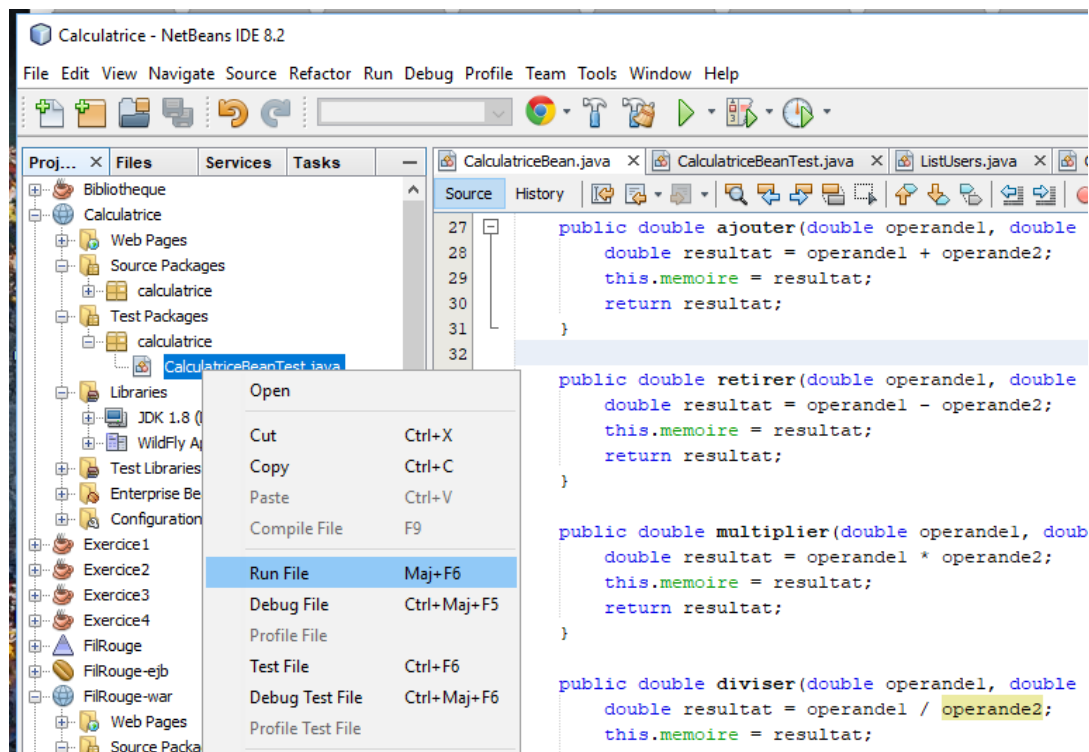
Si si, c'est important !

Exercice 7 : un simple EJB Session

```
@Test
public void testAjouter() throws Exception {
    double operande1 = 18.5;
    double operande2 = 17.5;
    EJBContainer container = javax.ejb.embeddable.EJBContainer.createEJBContainer();
    CalculatriceBean instance =
(CalculatriceBean)container.getContext().lookup("java:global/classes/CalculatriceBean");
    double expResult = 0.0;
    double result = instance.ajouter(operande1, operande2);
    assertEquals(expResult, result, 0.01);
    container.close() ;
}
```

Exercice 7 : un simple EJB Session

On lance ensuite le test unitaire.



Exercice 7 : un simple EJB Session

The screenshot displays an IDE window with a Java test class and its execution results. The test class, `CalculatriceBeanTest`, contains a single test method `testAjouter()` that verifies the `ajouter` method of `CalculatriceBean`. The test fails due to an exception.

```
44  /*
45  @Test
46  public void testAjouter() throws Exception {
47      System.out.println("ajouter");
48      double operande1 = 18.5;
49      double operande2 = 17.5;
50
51      EJBContainer container = javax.ejb.embeddable.EJBContainer.createEJBContainer();
52      CalculatriceBean instance = (CalculatriceBean) container.getContext().lookup("java:global/classes/CalculatriceBean");
53      double expResult = 0.0;
54      double result = instance.ajouter(operande1, operande2);
55      assertEquals(expResult, result, 0.01);
56      container.close();
57  }
```

The test results pane shows the following summary:

- Tests passed: 0,00 %
- No test passed, 1 test failed, 3 tests caused an error. (61,167 s)
- calculatrice.CalculatriceBeanTest Failed
- testAjouter Failed: expected:<0.0> but was:<36.0>
- testMultiplier caused an ERROR: Cannot create a new embedded EJBContainer: previous
- testRetirer caused an ERROR: Cannot create a new embedded EJBContainer: previous
- testDiviser caused an ERROR: Cannot create a new embedded EJBContainer: previous

The console output shows the following messages:

```
WARN: WELD-000146: BeforeBeanDiscovery.addAnnotatedType(AnnotatedType<?>) use
mai 20, 2018 10:06:37 AM org.jboss.weld.bootstrap.events.BeforeBeanDiscoveryI:
WARN: WELD-000146: BeforeBeanDiscovery.addAnnotatedType(AnnotatedType<?>) use
mai 20, 2018 10:06:37 AM org.jboss.weld.bootstrap.events.BeforeBeanDiscoveryI:
WARN: WELD-000146: BeforeBeanDiscovery.addAnnotatedType(AnnotatedType<?>) use
mai 20, 2018 10:06:37 AM org.jboss.weld.bootstrap.events.BeforeBeanDiscoveryI:
WARN: WELD-000146: BeforeBeanDiscovery.addAnnotatedType(AnnotatedType<?>) use
mai 20, 2018 10:06:37 AM org.jboss.weld.bootstrap.events.BeforeBeanDiscoveryI:
WARN: WELD-000146: BeforeBeanDiscovery.addAnnotatedType(AnnotatedType<?>) use
mai 20, 2018 10:06:38 AM org.glassfish.deployment.admin.DeployCommand execute
INFOS: classes was successfully deployed in 30 573 milliseconds.
```

Exercice 7 : un simple EJB Session

Pourquoi les tests qui suivent `testAjouter()` sont-ils indiqués en erreur ??

Exercice 7 : un simple EJB Session

Comment pourrait-on résoudre ce problème :

1. par une structure du langage Java ?
2. mieux : en améliorant la conception du test ?

Exercice 7 : un simple EJB Session

Implémenter et tester la calculatrice.

Exercice 8 : Une calculatrice avec mémoire

Exercice 8 : la calculatrice améliorée

On veut améliorer la calculette, de manière que chaque opération enregistre le résultat du dernier calcul dans une mémoire, et permette de récupérer ce résultat plus tard.

On veut de plus que les calculs puissent être réalisés via un navigateur.

Pour l'instant, on conserve le même EJB, on ne fait que lui ajouter un attribut privé pour représenter la mémoire, et on lui ajoute également une méthode publique permettant de récupérer le contenu de cet attribut.

Exercice 8 : la calculatrice améliorée

Pour ce qui est de l'accès par Internet à l'EJB, comment va t'on faire ?

- Que représente le navigateur dans une architecture n-tier ?
- Que nous faut-il d'autre ?

Exercice 8 : la calculatrice améliorée

Le navigateur représente un client Web.

Il nous faut aussi un composant de la couche Web pour faire le lien entre le client Web et notre EJB.

⇒ Un composant du type servlet, JSP ou JSF

Ceci sera présenté par Sébastien semaine prochaine

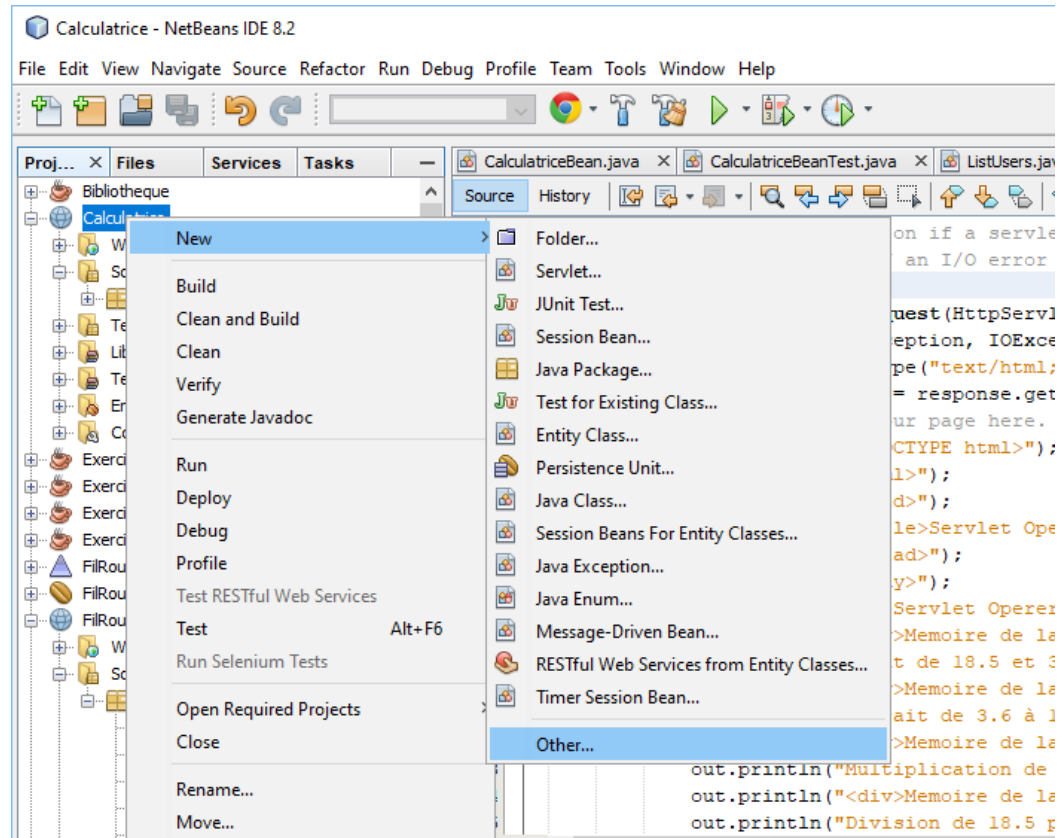
⇒ on va utiliser une simple servlet sans explication

Exercice 8 : la calculatrice améliorée

1. Modifier CalculatriceBean pour lui ajouter une mémoire et un moyen de lire et d'écrire dans la mémoire.
2. Créer une servlet comme indiqué ci-après.

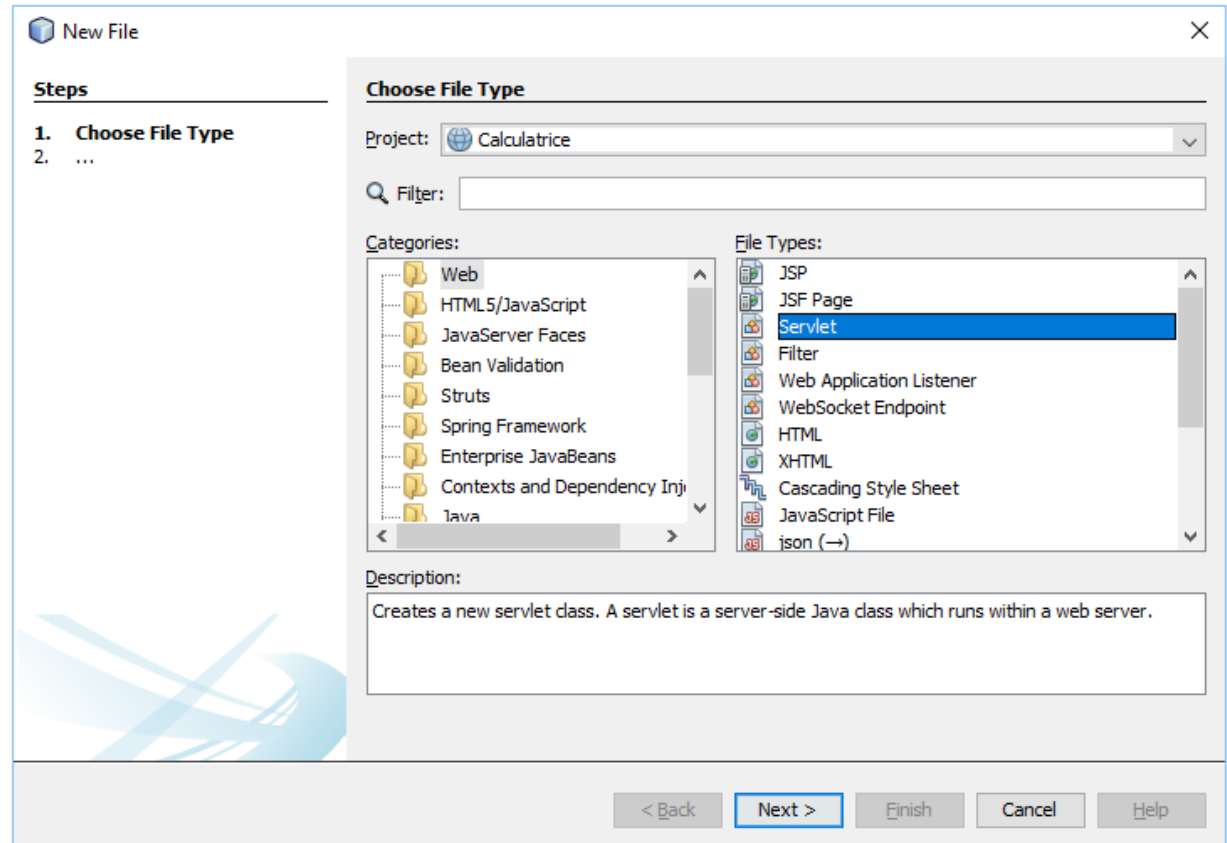
Exercice 8 : la calculatrice améliorée

Création d'une servlet (étape 1)



Exercice 8 : la calculatrice améliorée

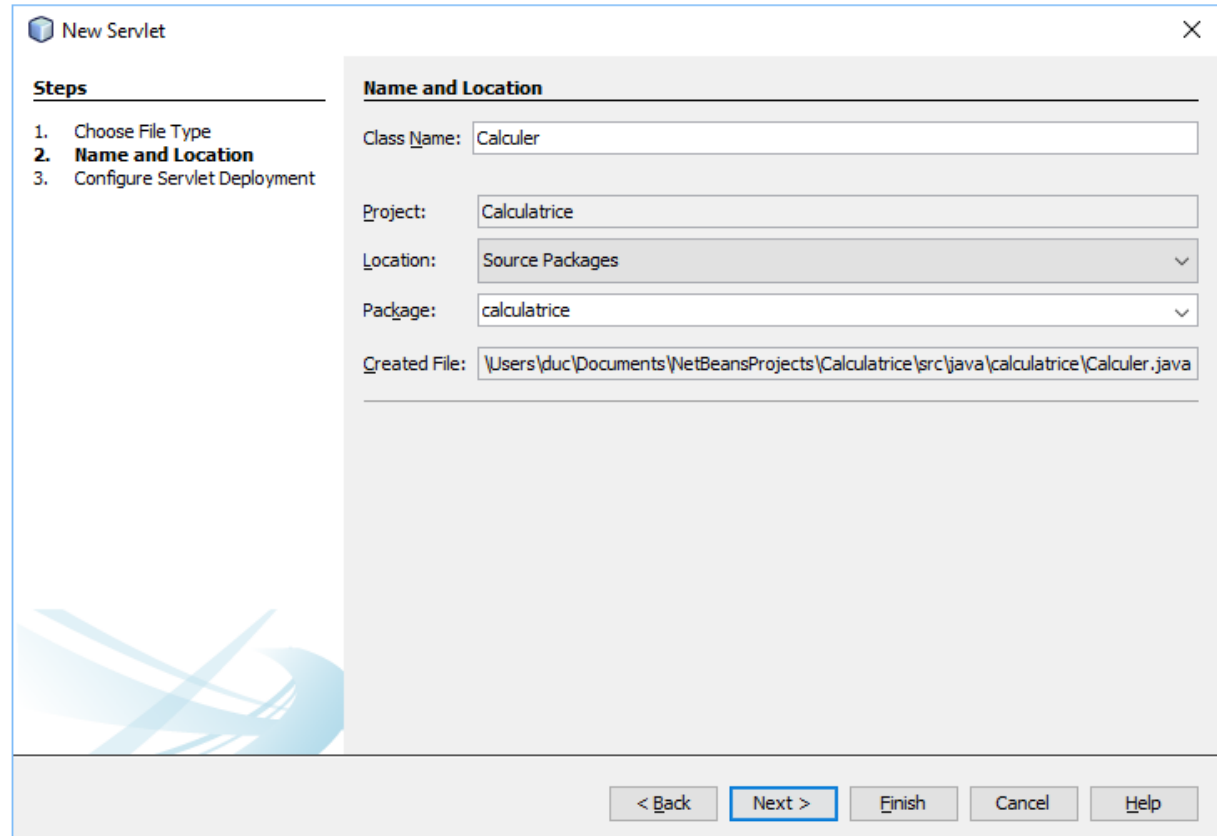
Création d'une servlet (étape 2)



Exercice 8 : la calculatrice améliorée

Création d'une servlet (étape 3)

Attention :
mentionner le
même package
que pour l'EJB
(calculatrice).



The screenshot shows the 'New Servlet' dialog box with the 'Name and Location' tab selected. The 'Steps' list on the left indicates the current step is '2. Name and Location'. The 'Class Name' is 'Calculer'. The 'Project' is 'Calculatrice'. The 'Location' is 'Source Packages'. The 'Package' is 'calculatrice'. The 'Created File' path is shown as '\\Users\\duc\\Documents\\NetBeansProjects\\Calculatrice\\src\\java\\calculatrice\\Calculer.java'. The 'Next >' button is highlighted.

New Servlet

Steps

1. Choose File Type
- 2. Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

< Back Next > Finish Cancel Help

Exercice 8 : la calculatrice améliorée

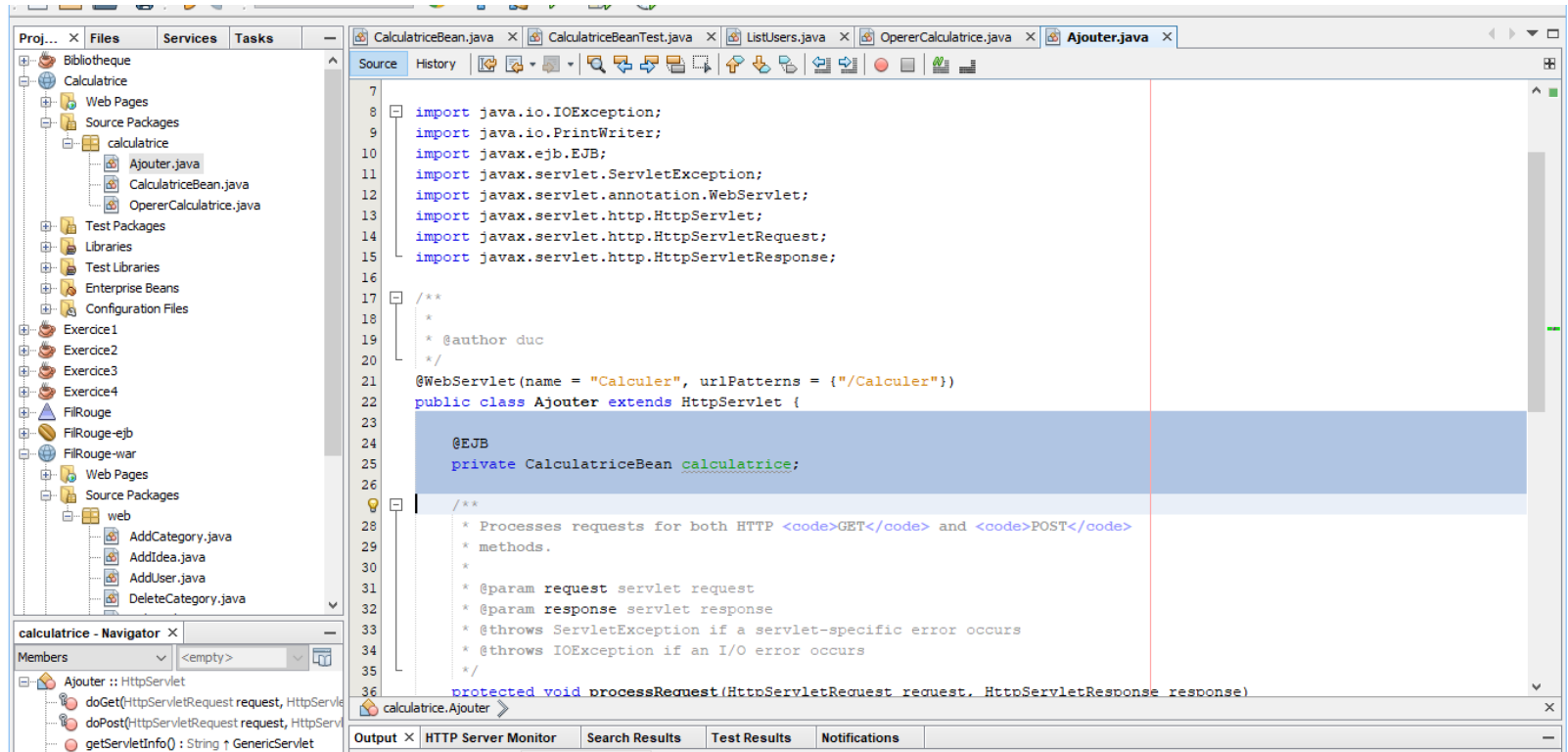
On injecte dans cette servlet l'EJB Session correspondant à la calculatrice :

```
@EJB
```

```
private CalculatriceBean calculatrice;
```

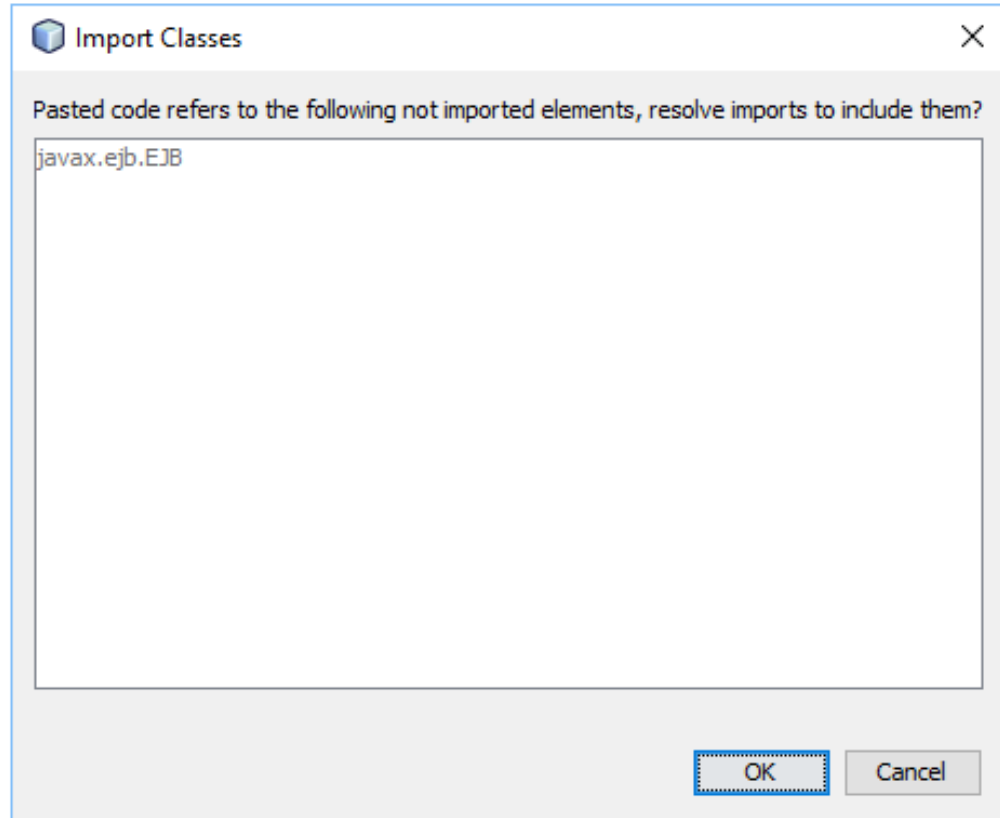
Cette injection permet de définir un attribut de la servlet qui sera du type `CalculatriceBean` et sera automatiquement instancié par le conteneur d'EJB.

Exercice 8 : la calculatrice améliorée



Exercice 8 : la calculatrice améliorée

NetBeans vous propose d'importer les dépendances nécessaires



Exercice 8 : la calculatrice améliorée

On insère dans la méthode `processRequest()` de cette servlet le code qui utilise la calculatrice pour effectuer des calculs et afficher sa mémoire après chaque calcul.

NB. Les opérandes des opérations sont hardcodées pour ne pas rentrer dans les détails du passage de paramètres de requêtes HTTP par le client et leur traitement dans la servlet, vous verrez ça avec Sébastien...

Exercice 8 : la calculatrice améliorée

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    out.println("<div>Memoire de la calculatrice : " +
calculatrice.getMemoire()+ "</div>");

    out.println("Ajout de 18.5 et 3.6 : " + calculatrice.ajouter(18.5, 3.6));

    out.println("<div>Memoire de la calculatrice : " +
calculatrice.getMemoire()+ "</div>");

    out.println("Retrait de 3.6 à 18.5 : " + calculatrice.retirer(18.5,
3.6));

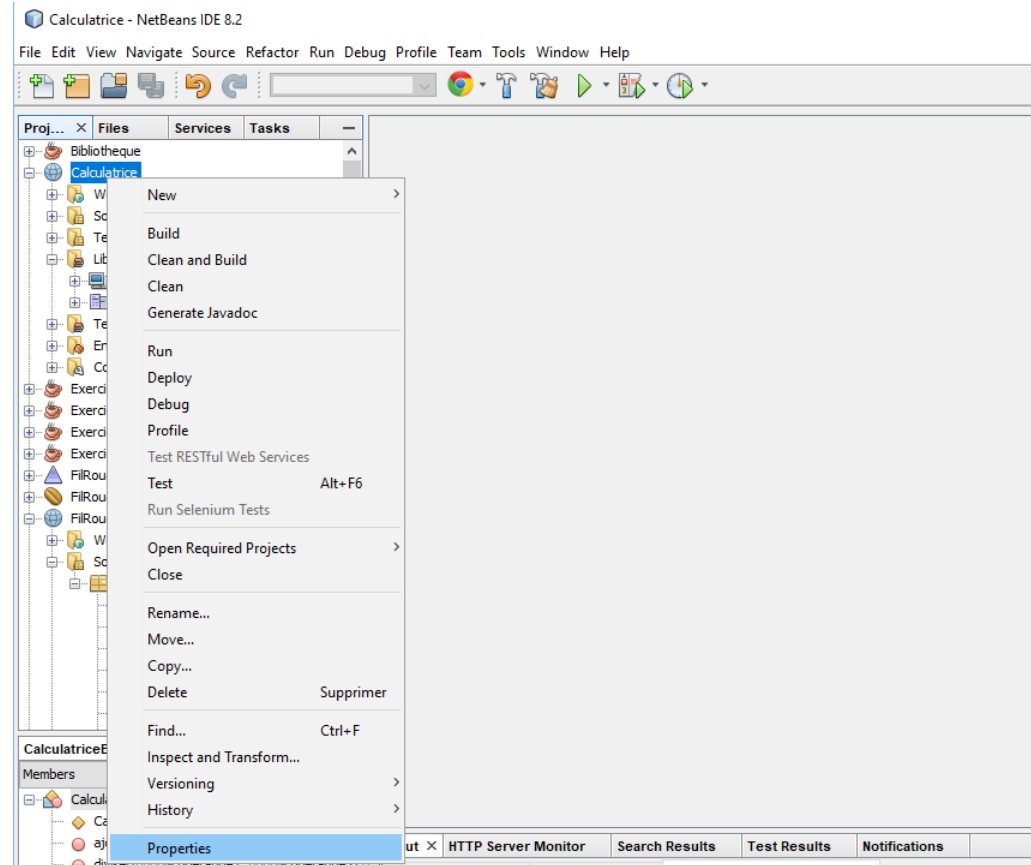
    out.println("<div>Memoire de la calculatrice : " +
calculatrice.getMemoire()+ "</div>");

    ...
}
```

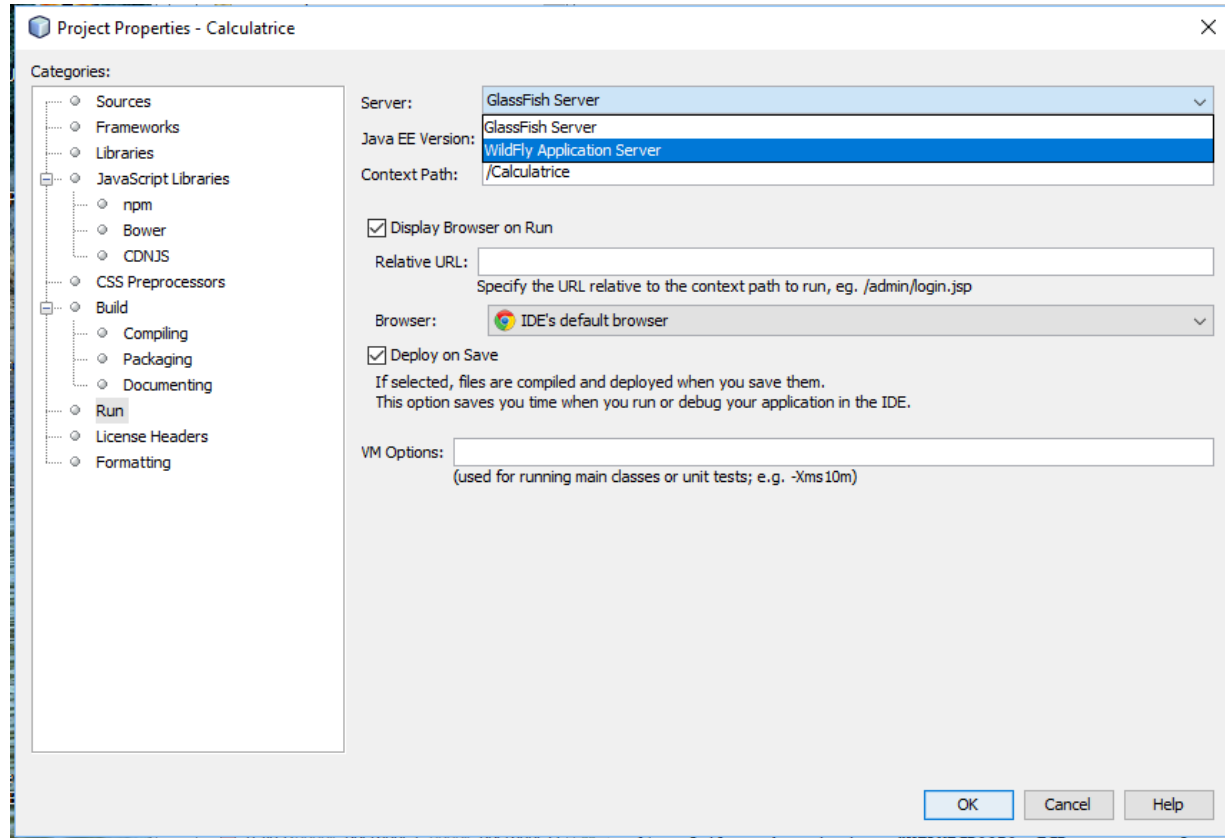
Exercice 8 : la calculatrice améliorée

On change de serveur d'application : on passe de GlassFish à WildFly.

Exercice 8 : la calculatrice améliorée



Exercice 8 : la calculatrice améliorée

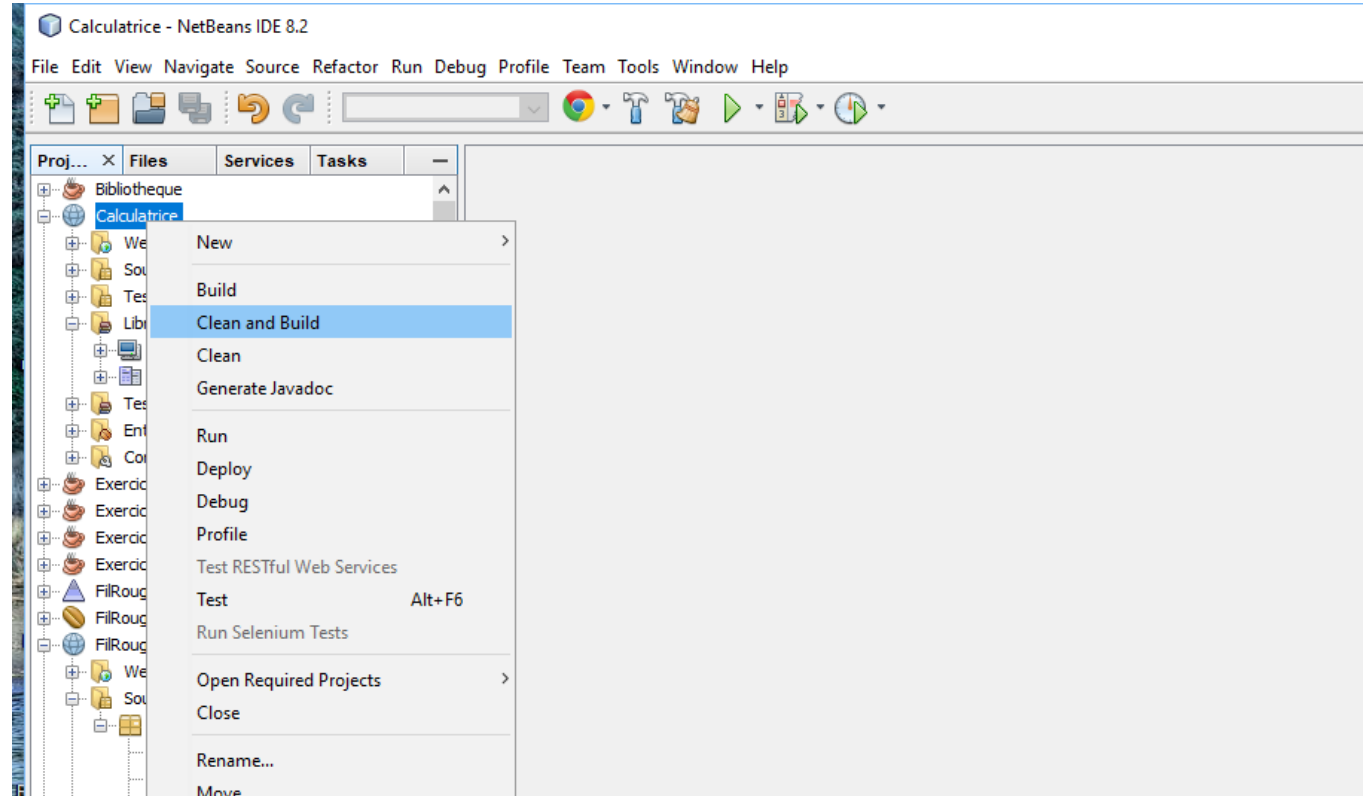


Exercice 8 : la calculatrice améliorée

On construit ensuite l'application Web et on la déploie sur le serveur d'application WildFly.

Exercice 8 : la calculatrice améliorée

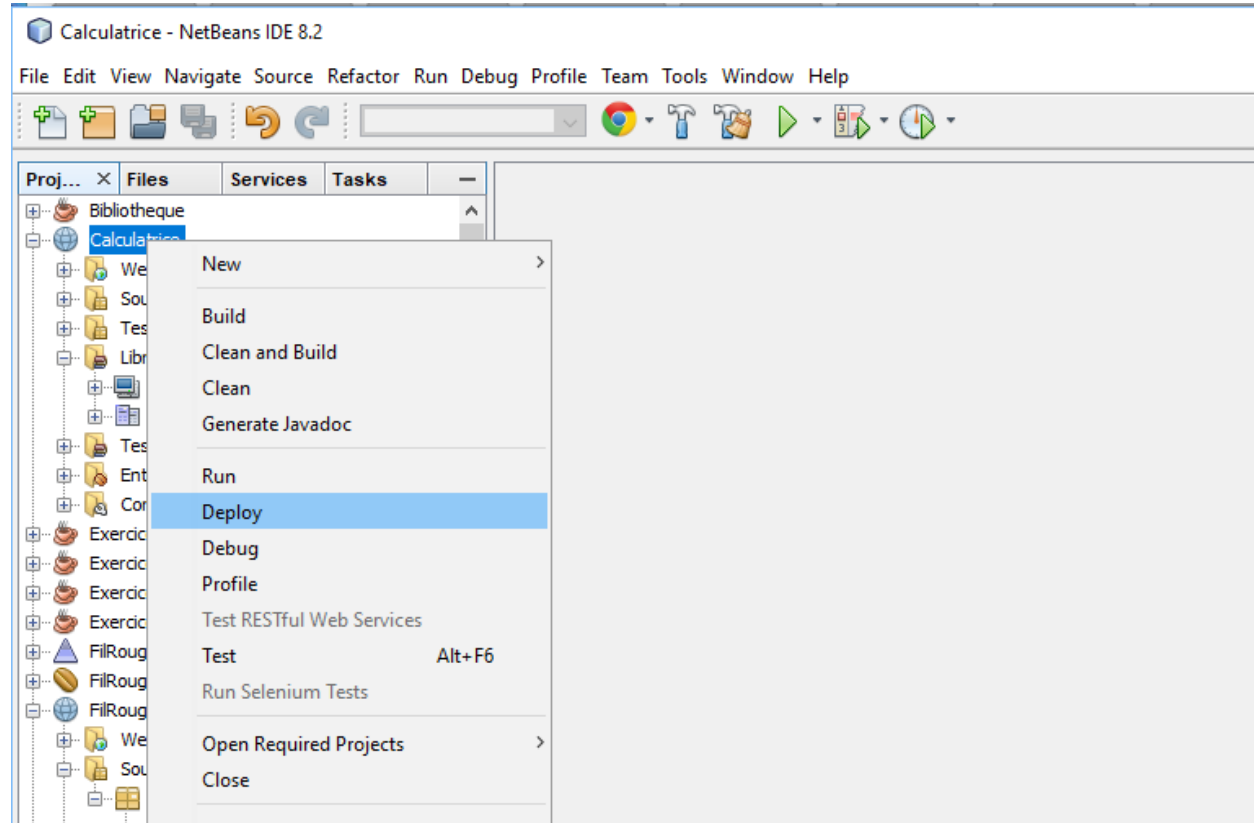
Construction
de l'application
Web



Exercice 8 : la calculatrice améliorée

Déploiement de l'application Web

NB. Ce déploiement peut être effectué automatiquement par l'IDE sur simple sauvegarde du source Java des composants.



Exercice 8 : la calculatrice améliorée



Problème possible : quelle peut être son origine ?

```
Output × HTTP Server Monitor Search Results Test Results Notifications
GlassFish Server × WildFly Application Server × Calculatrice (run-deploy) ×
ant -f E:\Users\duc\Documents\NetBeansProjects\Calculatrice -Dnb.internal.action.name=redeploy -Ddirectory.deployment.supported=true -DforceRedeploy=true -Dnb.wait.for.caches=true -Dbrow
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsp:
Starting WildFly Application Server
WildFly Application Server Start Failed. HTTP Connector port 8080 is already in use.
E:\Users\duc\Documents\NetBeansProjects\Calculatrice\nbproject\build-impl.xml:1045: Deployment error: WildFly Application Server Start Failed. HTTP Connector port 8080 is already in use.
See the server log for details.
BUILD FAILED (total time: 0 seconds)
```

Exercice 8 : la calculatrice améliorée

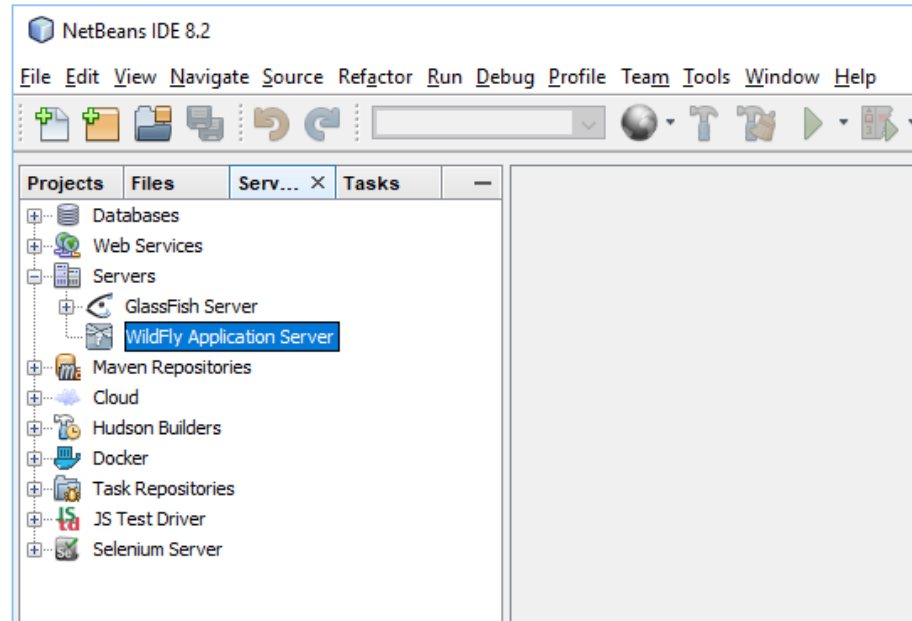
Normalement, un serveur d'application accepte le déploiement à chaud d'une application Web ou d'une application EJB.

Néanmoins, l'intégration entre un IDE et un serveur d'application n'est pas parfaite, et le déploiement peut demander le lancement du serveur d'application.

Sauf que... un serveur d'application peut déjà être en train de s'exécuter. Il occupe donc le port HTTP standard qu'utilisent les serveurs d'application : **8080**

Exercice 8 : la calculatrice améliorée

Il faut donc arrêter manuellement le serveur d'application, en se rendant dans l'onglet dans NetBeans relatifs aux serveurs et autres ressources.

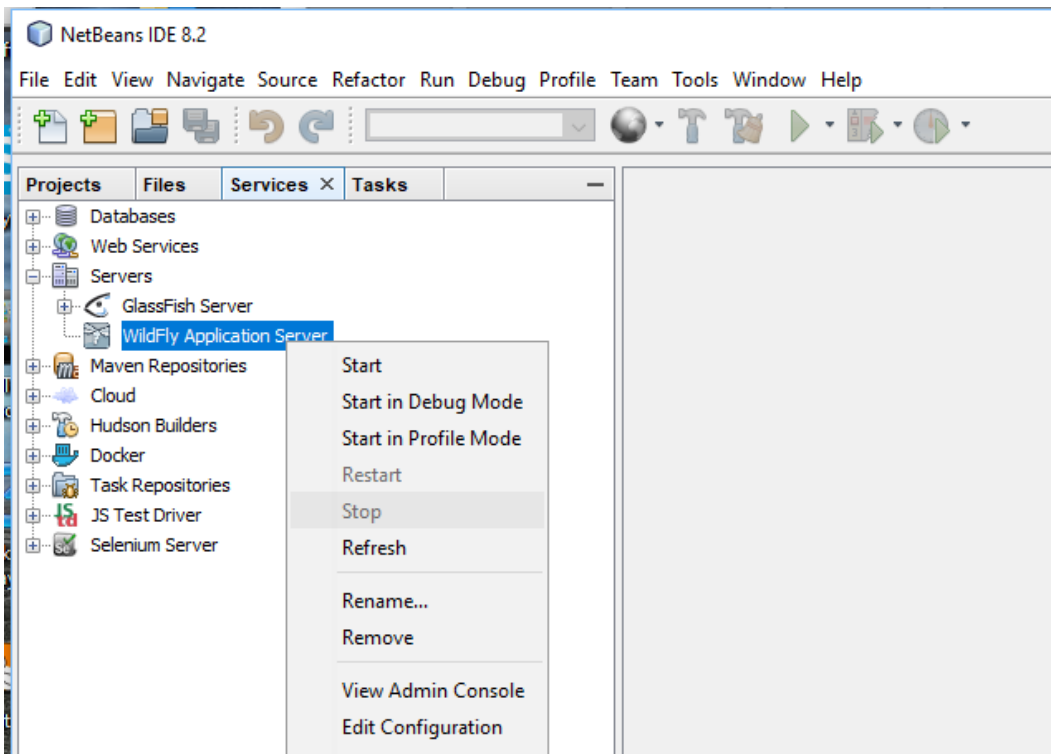


Exercice 8 : la calculatrice améliorée

Sauf que, là aussi...

L'intégration de WildFly version 12.0 dans NetBeans n'est pas parfaite.

Seule solution : arrêter NetBeans et le redémarrer.



Exercice 8 : la calculatrice améliorée

Concernant l'intégration entre WildFly version 12.0 et NetBeans version 8.2 :

- il est recommandé de lancer WildFly à la main (via un shell Linux ou une fenêtre `cmd.exe` sous Windows) et de déployer les applications Web (suffixées par `.war`) ou JEE (suffixées par `.ear`) à la main
- ensuite, il est possible de déployer une application Web ou JEE par simple copie d'un fichier `.war` ou `.ear` dans le répertoire de déploiement de WildFly :

`<racine de WildFly>\standalone\deployments`

Par exemple :

`E:\wildfly-12.0.0.Final\standalone\deployments`

Exercice 8 : la calculatrice améliorée

En pratique, cela marche facilement pour le premier déploiement, ou si l'application a été effacée, mais le remplacement d'une version par une autre à la volée ainsi ne fonctionne pas toujours.

Il est possible d'effacer le répertoire et les fichiers relatifs à une application Web ou JEE dans le répertoire de déploiement, mais la pratique la plus sûre consiste à passer par l'IHM d'administration de WildFly.

C'est une application Web accessible via le port **9990** sur `localhost`.

Exercice 8 : la calculatrice améliorée

Gestion de WildFly via son IHM d'administration

The screenshot shows the WildFly Management console in a web browser. The browser tab is titled 'WildFly Management' and the address bar shows 'localhost:9990/console/App.html#home'. The console has a dark header with the 'WildFly' logo and a navigation menu with links: Home, Deployments, Configuration, Runtime, Access Control, and Patching. The main content area is divided into four sections, each with an icon, a title, a description, and a 'Start' button.

- Deployments** (Icon: Package): Add and manage deployments.
 - Deploy an application to the server.
 1. Use the 'New Deployment' wizard to deploy the application
 2. Enable the deployment
- Configuration** (Icon: Gears): Configure subsystem settings.
 - Create a Datasource
 - Define a datasource to be used by deployed application and registered.
 1. Select the Datasources subsystem
 2. Add an XA or non-XA datasource
 3. Use the 'Create Datasource' wizard to configure the
 - Create a JMS Queue
- Runtime** (Icon: Server): Monitor server status.
 - Monitor the Server
 - View runtime information such as server status, JVM status, and server log files.
- Access Control** (Icon: Users): Manage user and group permissions for manager.
 - Assign User Roles
 - Assign roles to users or groups to determine access

Exercice 8 : la calculatrice améliorée

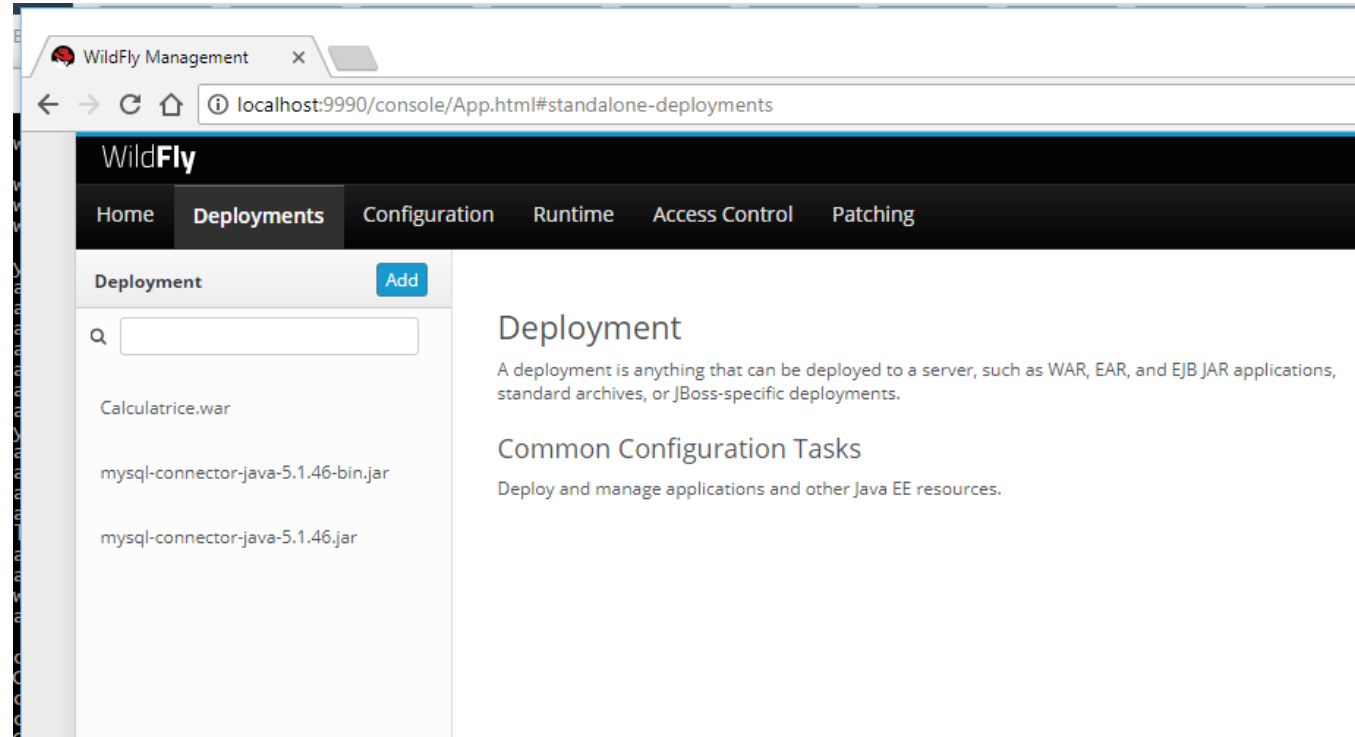
NB : C'est bien sûr l'IHM utilisée pour créer les datasources qui permettent d'accéder par driver JDBC ou JPA aux bases de données relationnelles.

Mode opératoire recommandé :

- supprimer l'application via l'IHM d'administration
- la déployer de nouveau via l'IHM d'administration en spécifiant le fichier `.war` ou `.ear`

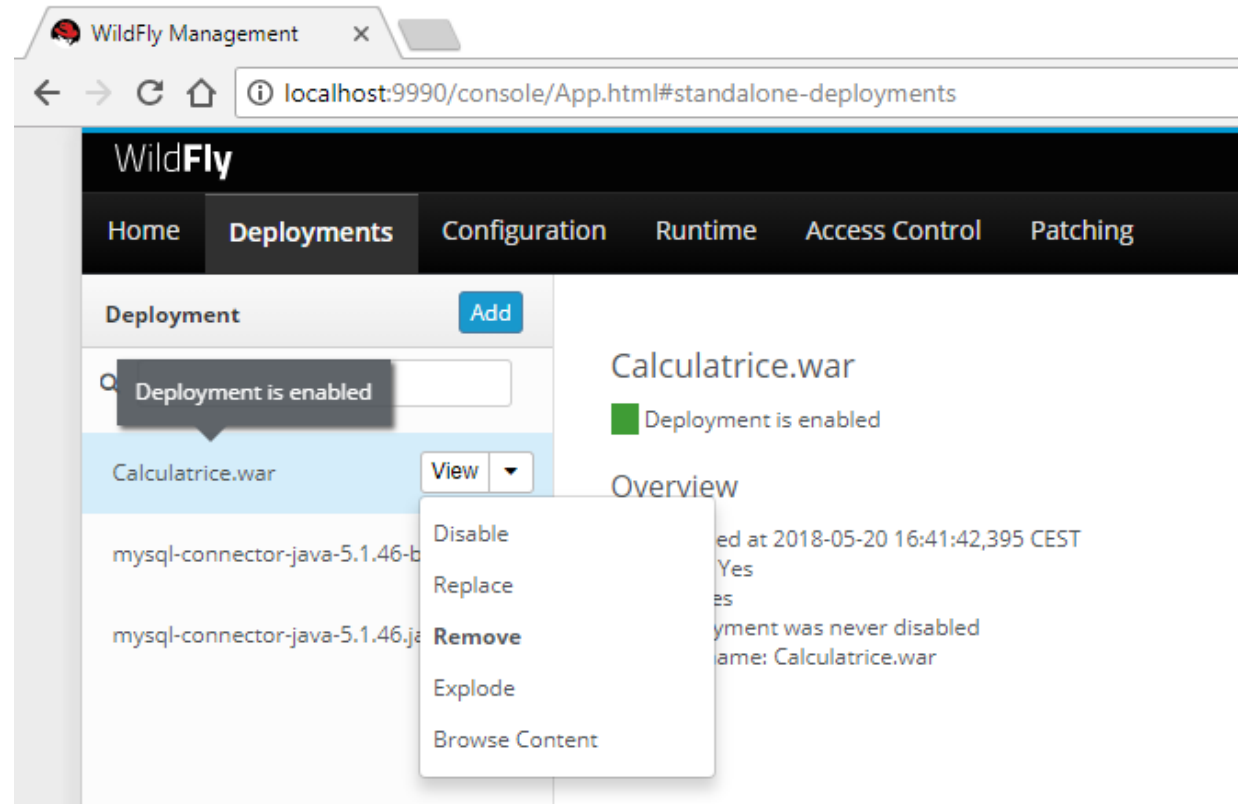
Exercice 8 : la calculatrice améliorée

Suppression
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 1)



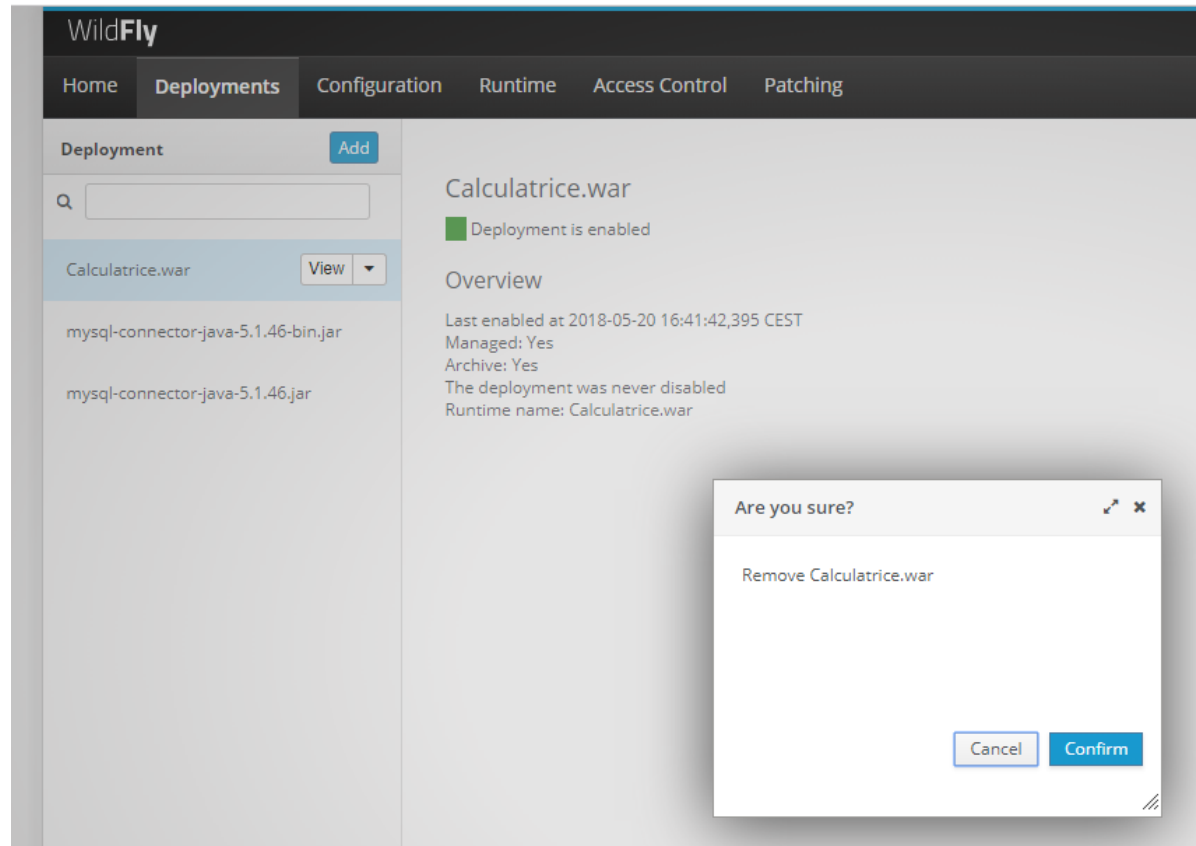
Exercice 8 : la calculatrice améliorée

Suppression
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 2)



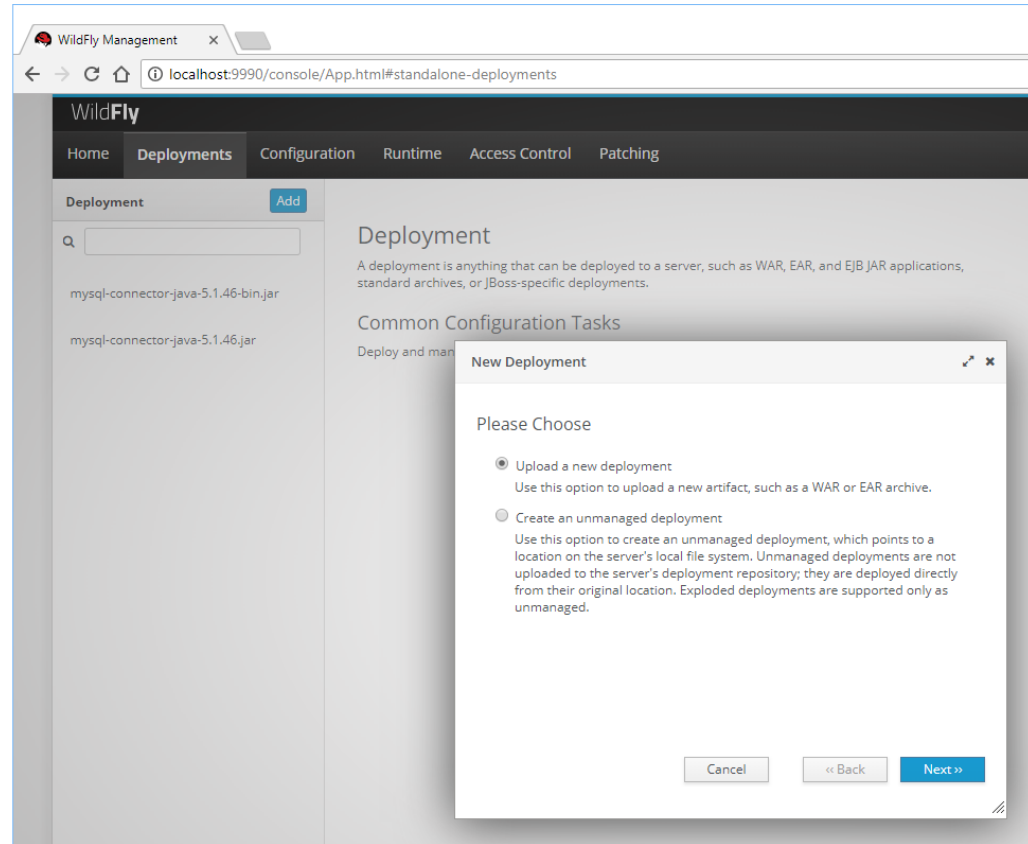
Exercice 8 : la calculatrice améliorée

Suppression
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 3)



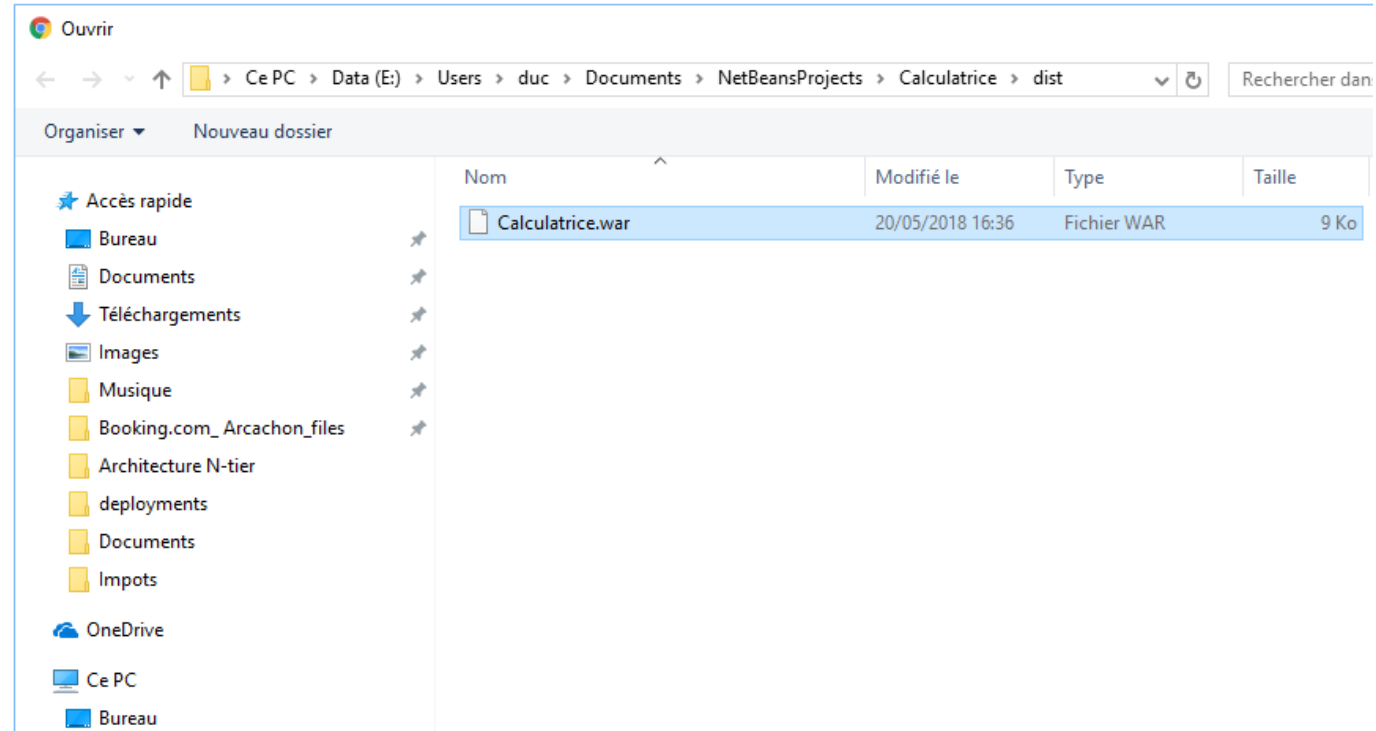
Exercice 8 : la calculatrice améliorée

Déploiement
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 1)



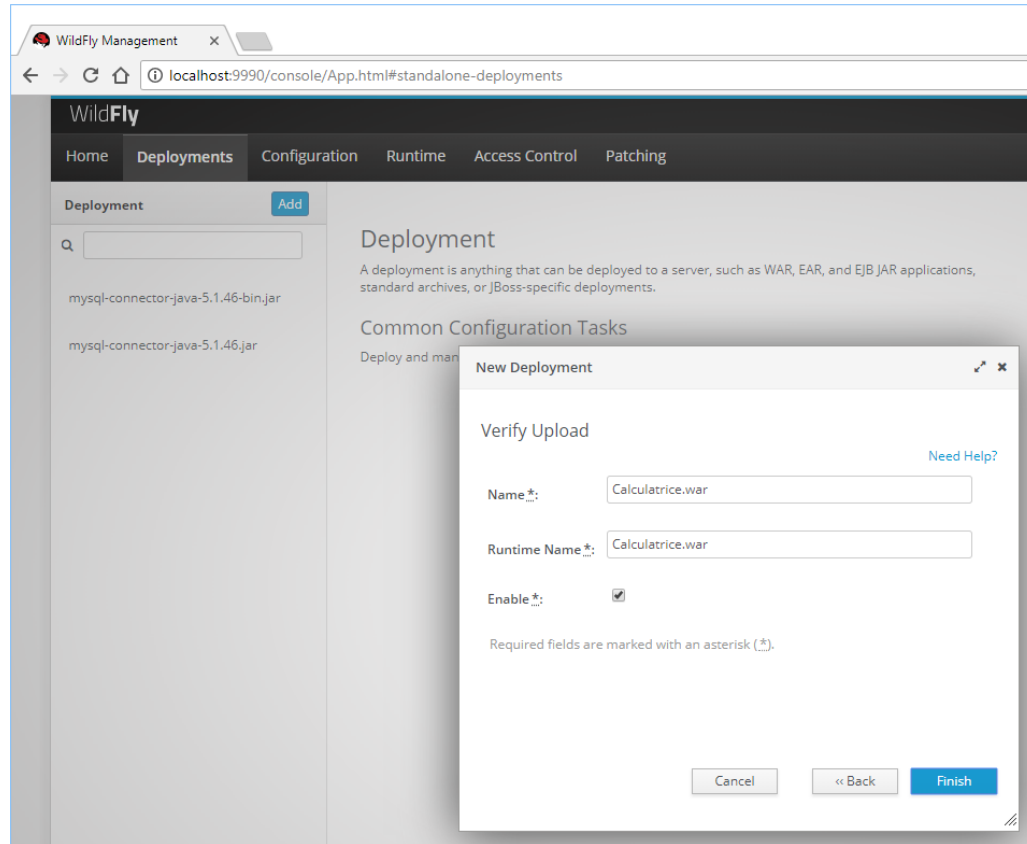
Exercice 8 : la calculatrice améliorée

Déploiement
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 2)



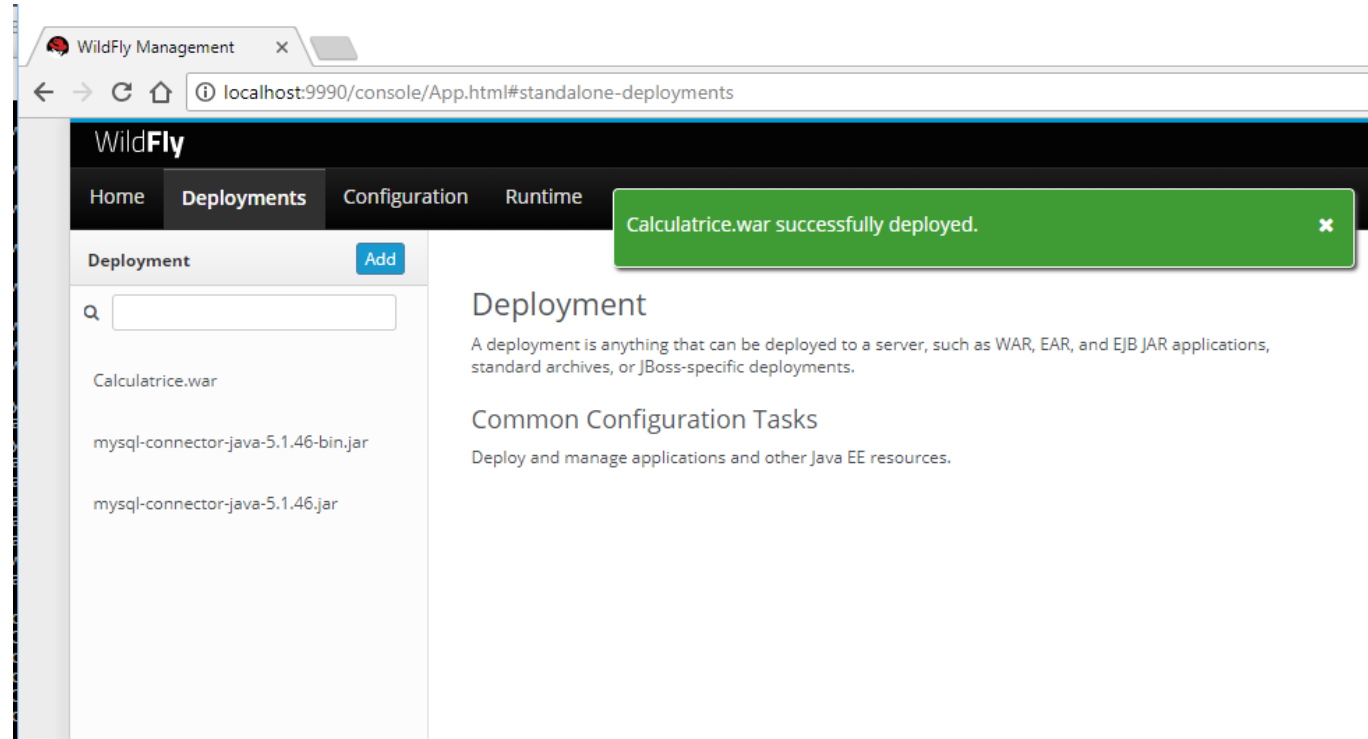
Exercice 8 : la calculatrice améliorée

Déploiement
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 3)



Exercice 8 : la calculatrice améliorée

Déploiement
d'une
application Web
depuis l'IHM
d'administration
de WildFly
(étape 4)



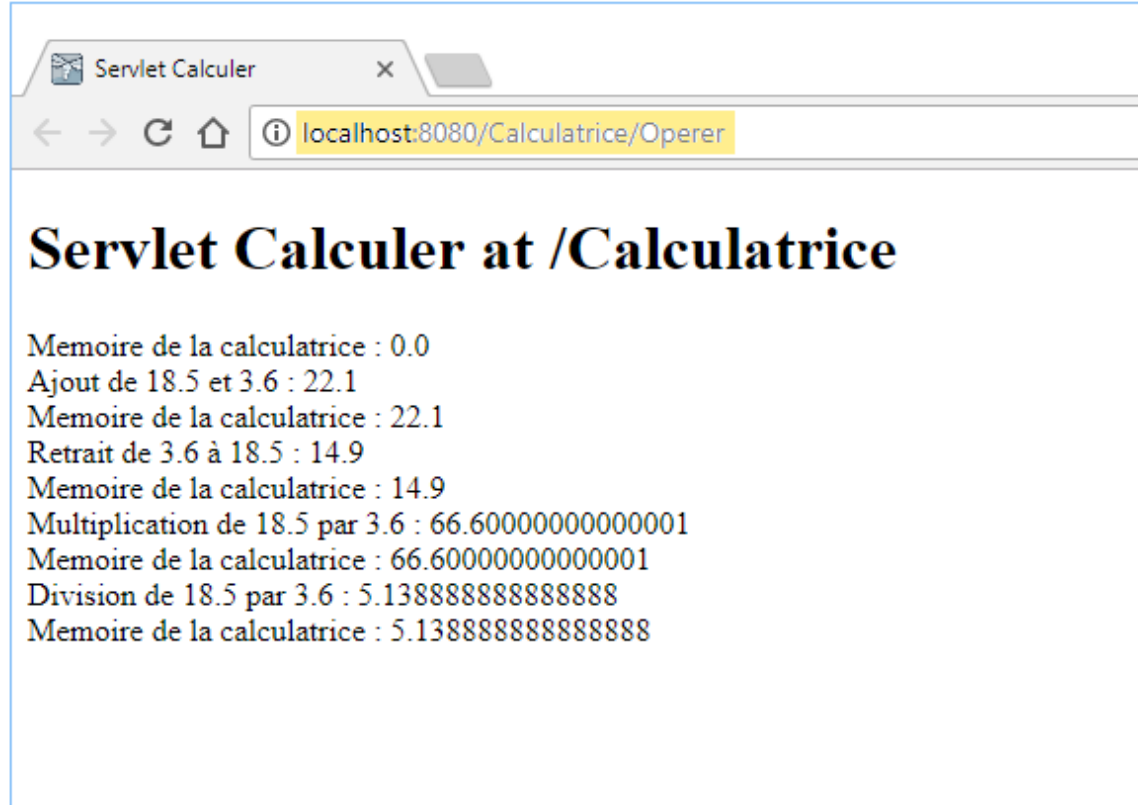
Exercice 8 : la calculatrice améliorée

Une fois l'application déployée, on peut y accéder via un navigateur Web.

Vous verrez plus tard comment est définie l'URL d'accès à la servlet.

Exercice 8 : la calculatrice améliorée

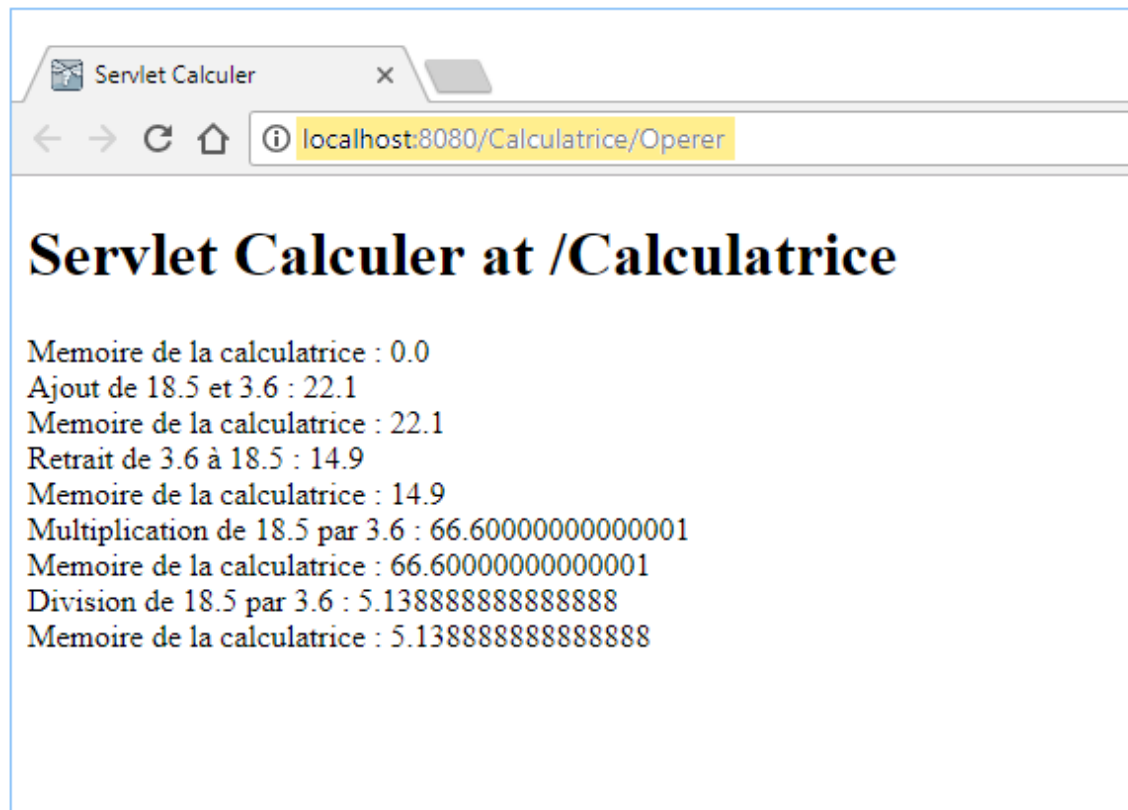
Une fois l'application déployée, on peut y accéder via un navigateur Web



Exercice 8 : la calculatrice améliorée

Mais... notre EJB
n'est-il pas **stateless** ?

Pourquoi la
calculatrice enregistre
t'elle correctement
dans sa mémoire le
dernier résultat ?



Exercice 8 : la calculatrice améliorée

Peut-être parce que toutes les opérations de calcul sont effectuées au cours du même échange HTTP ?

Il a été dit que « *l'enregistrement d'un état conversationnel s'étend sur tous les échanges entre un client et l'application* »

- tant que le serveur n'est pas arrêté puis redémarré
- ou qu'un timeout n'a pas été atteint.

Exercice 8 : la calculatrice améliorée

On va donc modifier l'application Web pour que les différentes opérations soient effectuées au cours de différents échanges HTTP : une servlet correspondra à une opération de la calculatrice.

A vous de jouer !

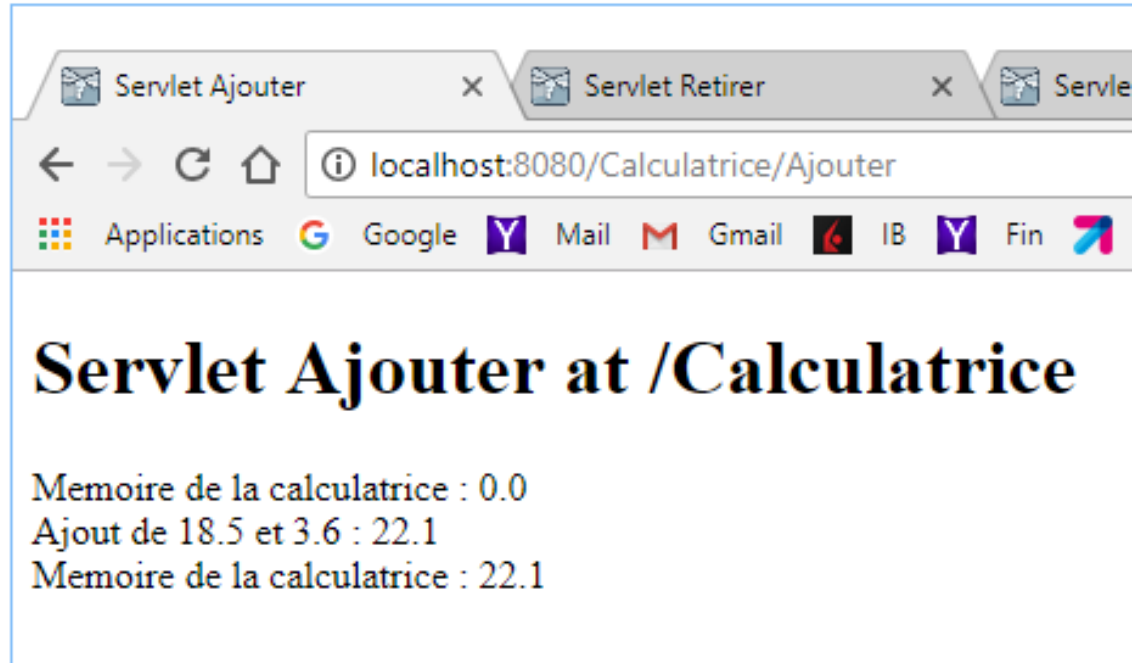
Exercice 8 : la calculatrice améliorée

Une fois l'application modifiée (ajout de 4 servlets, une par opération), on reconstruit l'application et on la redéploie.

Exercice 8 : la calculatrice améliorée

Une fois l'application redéployée, on peut y accéder de nouveau via un navigateur Web.

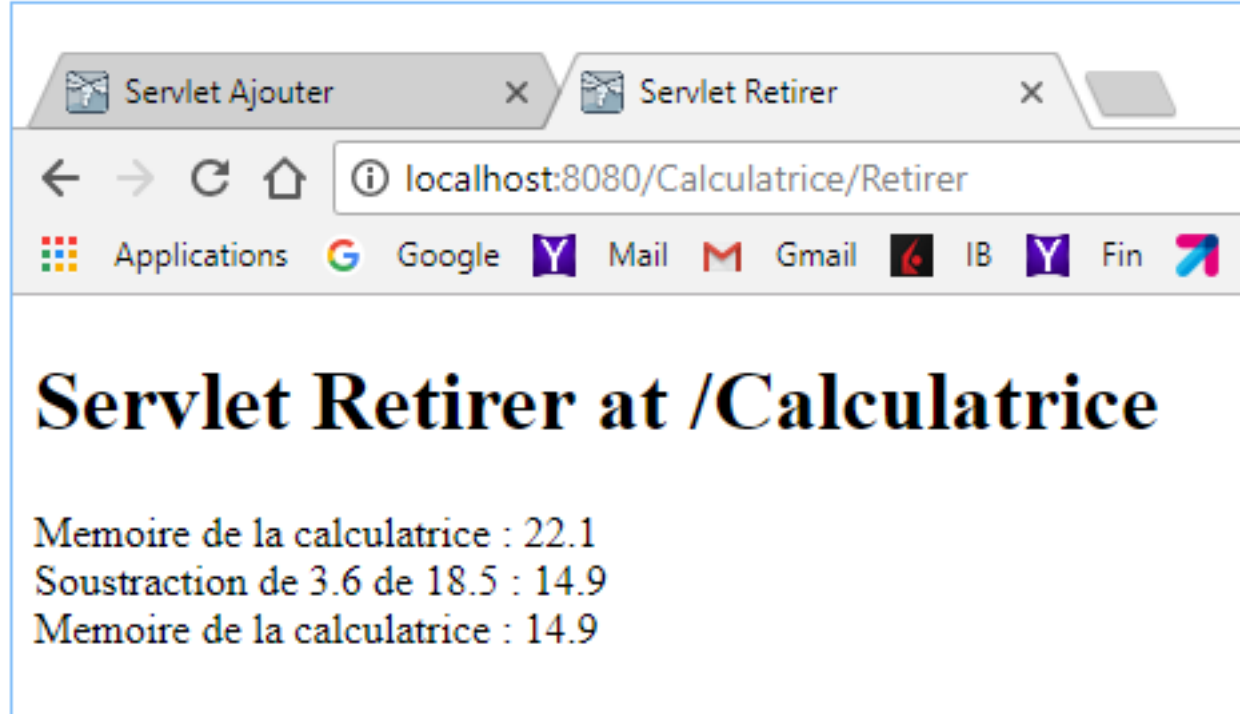
Ici, activation de l'opération d'addition.



Exercice 8 : la calculatrice améliorée

Ici, activation de l'opération de soustraction.

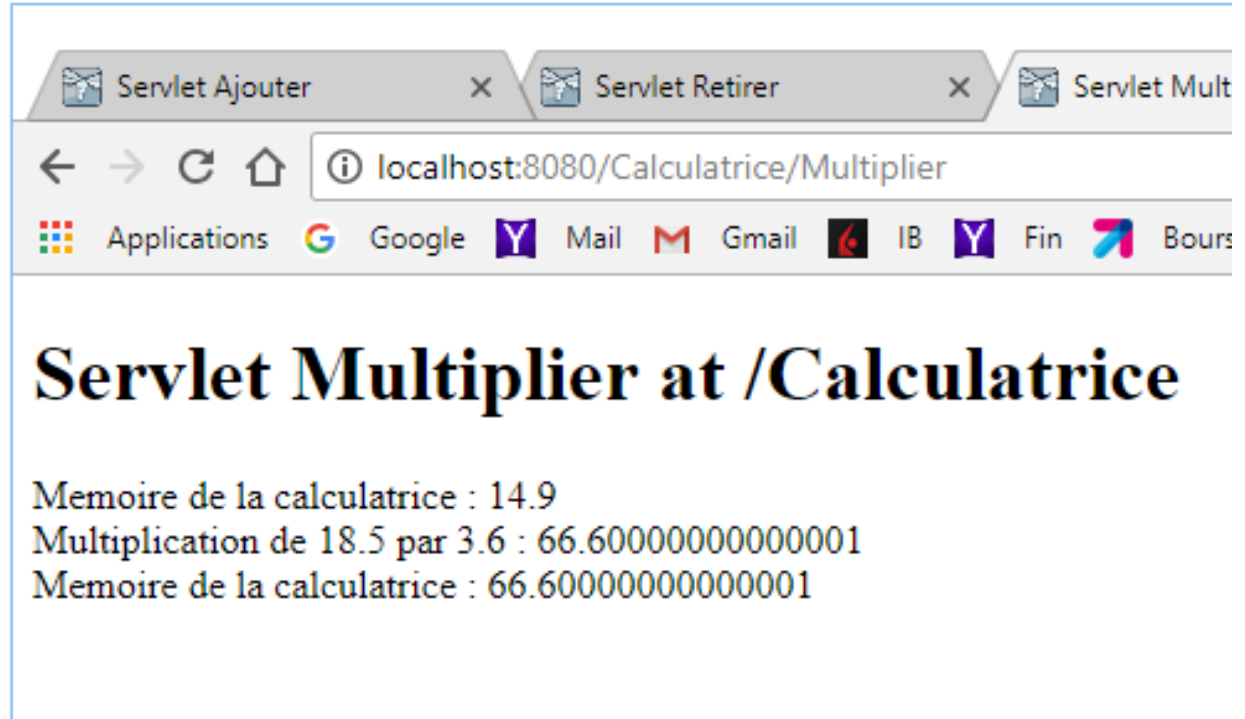
?????



Exercice 8 : la calculatrice améliorée

Ici, activation de l'opération de multiplication.

?????



Exercice 8 : la calculatrice améliorée

Pourquoi l'état de notre EJB est-il apparemment sauvegardé ?

Une instance particulière de l'EJB CalculatriceBean est pourtant accédée par chaque servlet...

En fait, rien ne garantit que l'instance ne soit pas la même : le conteneur d'EJB gère un **pool d'instance** et fournit une instance à la demande. Peut-être que ce sera la même, peut-être pas...

Dans ce cas de figure, c'est un problème conceptuel : si un EJB est dit stateless, il est **incorrect** (car risqué) de lui faire enregistrer un état interne.

Exercice 8 : la calculatrice améliorée

Par exemple, essayez d'activer les différents servlets à partir de différents clients Web (différents navigateurs plutôt que différents onglets du même navigateur) : vous vous rendrez peut-être compte que parfois la valeur en mémoire n'est pas la dernière calculée... Cela veut dire que le conteneur d'EJB a fourni une autre instance d'EJB que la dernière utilisée.

Exercice 8 : la calculatrice améliorée

La solution pour gérer correctement la mémoire de la calculatrice :

- si on veut que chaque client web voie sa propre valeur en mémoire, on peut peut-être transformer `CalculatriceBean` en EJB Session stateful ?
- si on veut que tous les clients web voient la même valeur en mémoire, alors on... on quoi ??

Exercice 8 : la calculatrice améliorée

Si on veut que tous les clients web voient la même valeur en mémoire, alors on fait de `CalculatriceBean` un singleton.

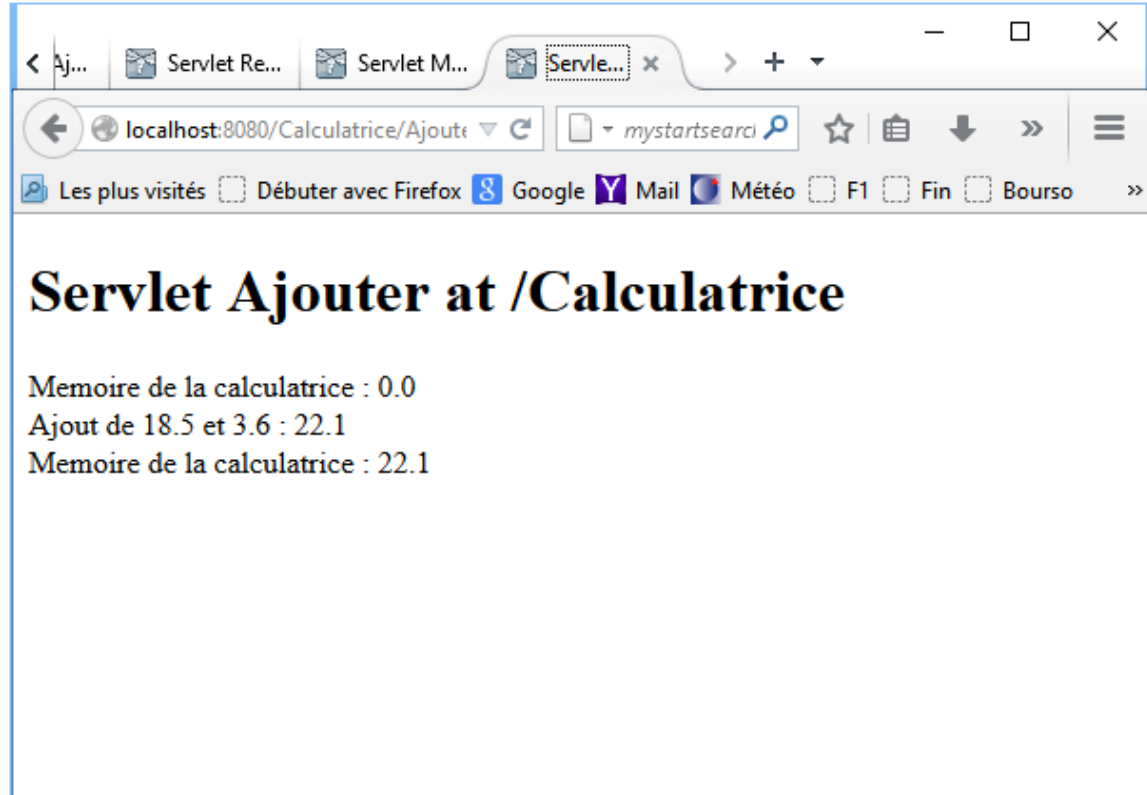
⇒ Implémentez la première approche !

Quelles sont les opérations compliquées à réaliser ?

Exercice 8 : la calculatrice améliorée

Une fois l'application modifiée et redéployée, on peut y accéder de nouveau via un navigateur Web.

Ici, activation de l'opération d'addition.

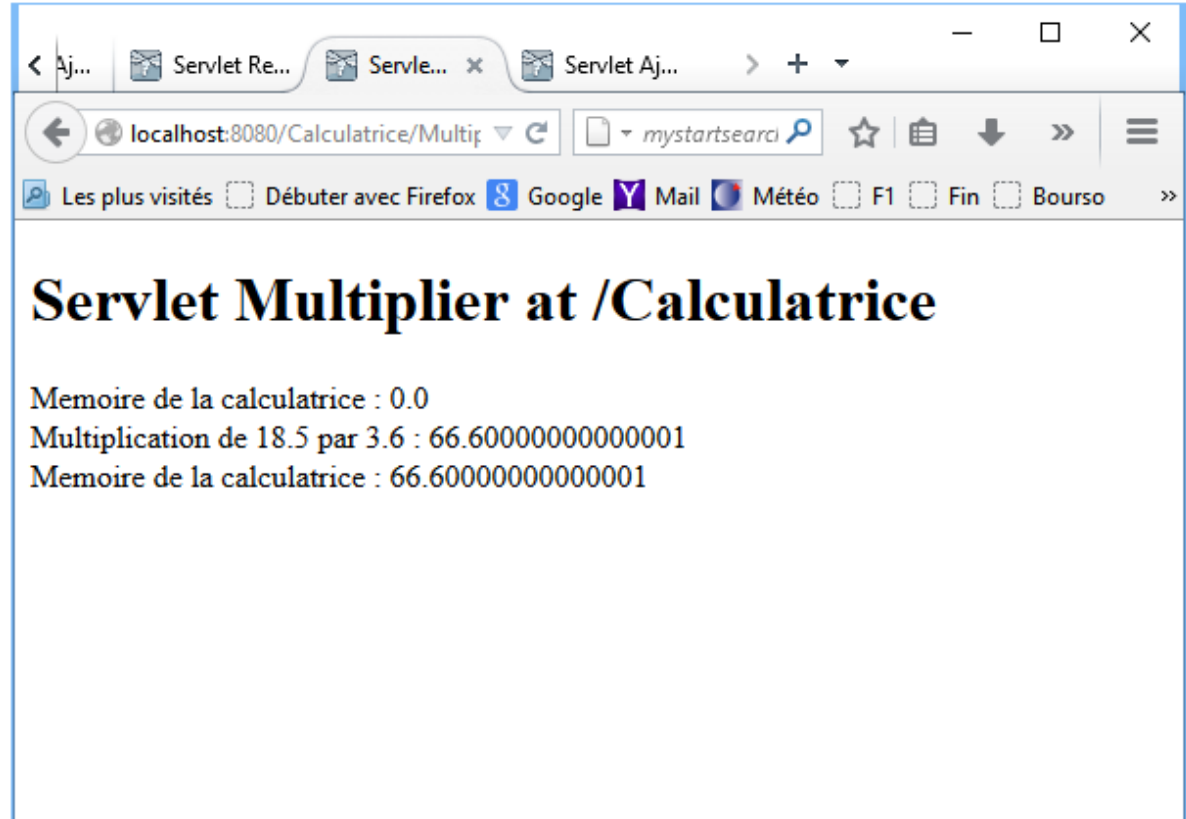


Exercice 8 : la calculatrice améliorée

Activation de
l'opération de
multiplication.

?????

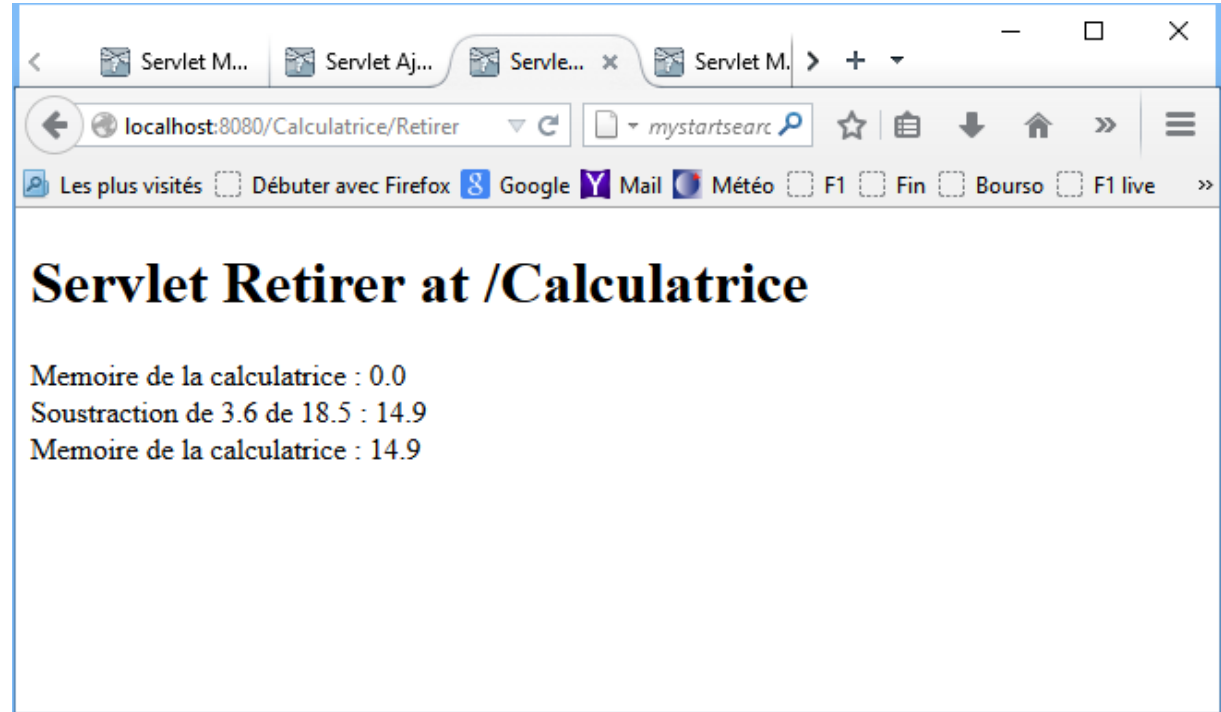
Pourquoi la
valeur de la
mémoire est-
elle incorrecte ?



Exercice 8 : la calculatrice améliorée

Activation de
l'opération de
soustraction.

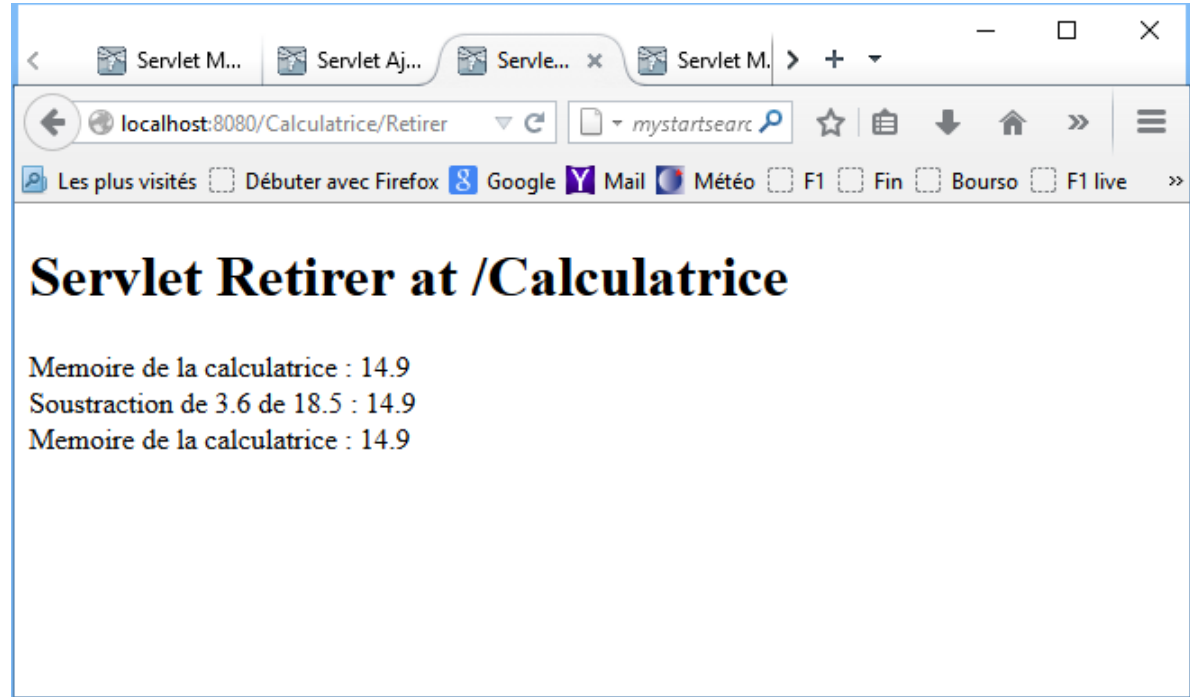
La
mémoire semble
être réinitialisée
à chaque fois...



Exercice 8 : la calculatrice améliorée

Nouvelle
activation de
l'opération de
soustraction.

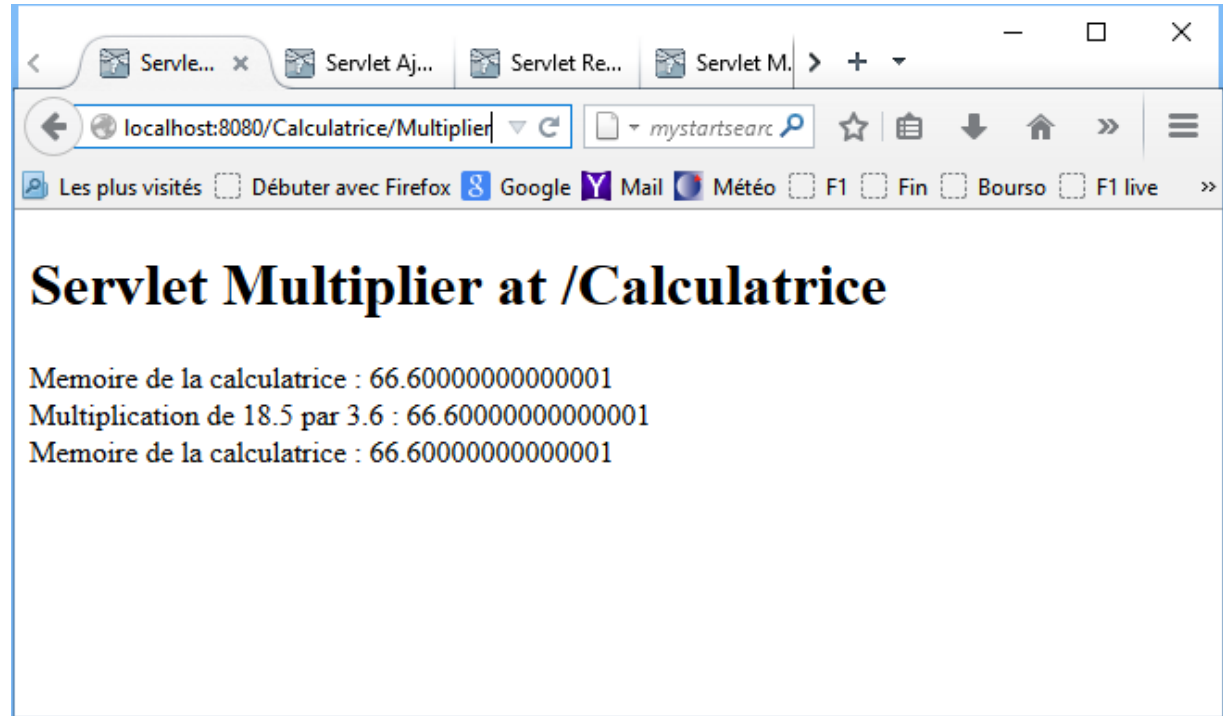
En fait, non.



Exercice 8 : la calculatrice améliorée

Nouvelle
activation de
l'opération de
multiplication.

La dernière
valeur calculée
est mémorisée,
mais servlet par
servlet.

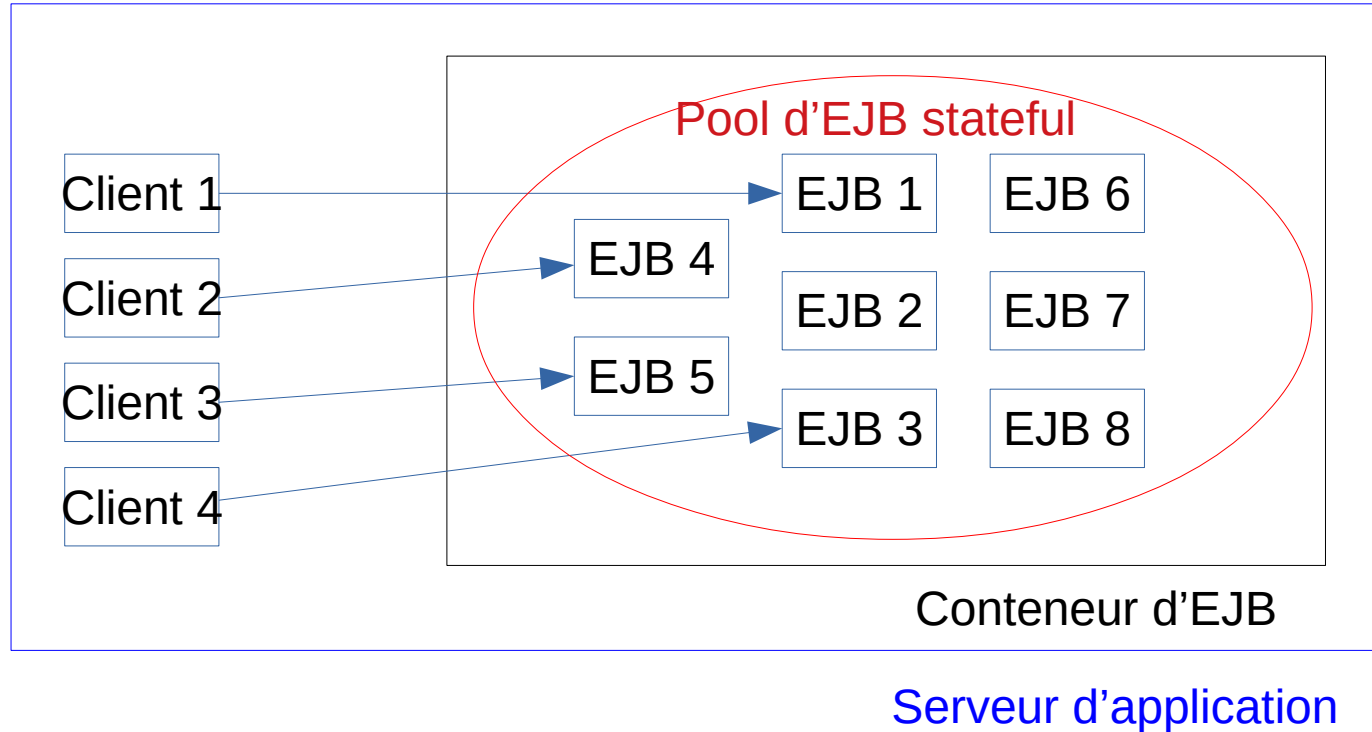


Exercice 8 : la calculatrice améliorée

L'annotation `@Stateful` nous garantit que « *d'un appel de méthode à l'autre, les attributs de l'instance d'EJB sont persistants* ».

Mais qui réalise les appels de méthode successifs – autrement dit, qui est le le **client** de l'EJB ? Ce qu'on appelle client n'est **pas** le client HTTP: c'est la servlet, c'est-à-dire le composant qui appelle directement l'EJB.

Exercice 8 : la calculatrice améliorée



Exercice 8 : la calculatrice améliorée

L'annotation `@Stateful` nous garantit aussi que l'instance d'EJB injectée dans une servlet sera toujours la même, du moins tant que le serveur d'application ne sera pas arrêté.

Du coup, la mémoire de l'instance d'EJB associée à chaque servlet est modifiée à chaque appel

⇒ on obtient donc une calculatrice qui possède une mémoire spécifique à chaque servlet (i.e. client)

Exercice 8 : la calculatrice améliorée

Dans le cas d'un EJB stateless, n'importe quelle instance d'EJB peut être utilisée – y compris la même, ce qui explique que l'invocation de chaque servlet peut modifier la mémoire de cette instance

⇒ on obtient une calculatrice qui possède une mémoire apparemment partagée par tous les clients HTTP et toutes les servlets

Exercice 9 : La gestion des utilisateurs

Exercice 9 : la gestion des utilisateurs

On veut qu'un utilisateur puisse se faire connaître du site Topaidi, par exemple quand l'identité de l'utilisateur est nécessaire :

- proposition d'une nouvelle idée
- vote pour une idée
- désactivation d'une idée
- validation d'un utilisateur par un administrateur
- etc.

Exercice 9 : la gestion des utilisateurs

L'utilisateur s'identifie au moyen de son adresse mail.

On se propose d'enregistrer dans la session HTTP une représentation de l'utilisateur, de manière à ce qu'un même utilisateur (navigateur Web) puisse effectuer toutes ses opérations sans devoir saisir son adresse mail à chaque fois.

Exercice 9 : la gestion des utilisateurs

L'application fournira quelques opérations :

- s'identifier
- créer une idée
- voter
- ...

Chaque opération ne fera qu'afficher un texte prédéfini indiquant l'identité de l'utilisateur, c'est-à-dire son adresse mail.

Exercice 9 : la gestion des utilisateurs

Pour simplifier les choses, on ne passera pas de paramètre à la servlet : l'utilisateur ne fournira pas d'adresse mail, une adresse mail aléatoire sera générée.

Implémenter la gestion des utilisateurs.

Exercice 10 : Une façade pour les idées

Exercice 10 : une façade pour les idées

On va implémenter une façade fournissant à la couche Web des services d'accès à l'entité `IdeeEntity`.

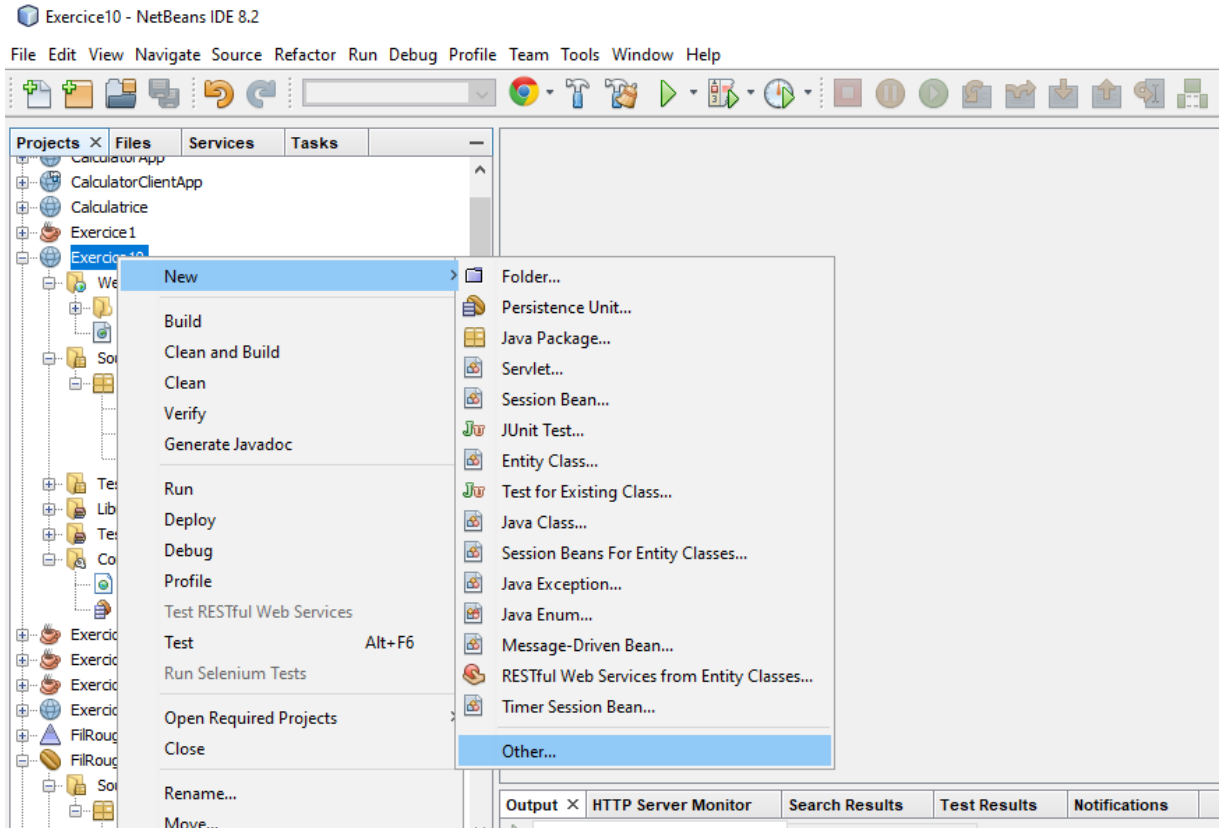
Cette façade sera testée au moyen de JUnit.

On crée un projet de type Java Web / Web Application, et on choisit GlassFish comme serveur d'application.

Si elle est définie, on copie l'entité `Idee` définie dans un projet NetBeans précédent à partir du projet correspondant.

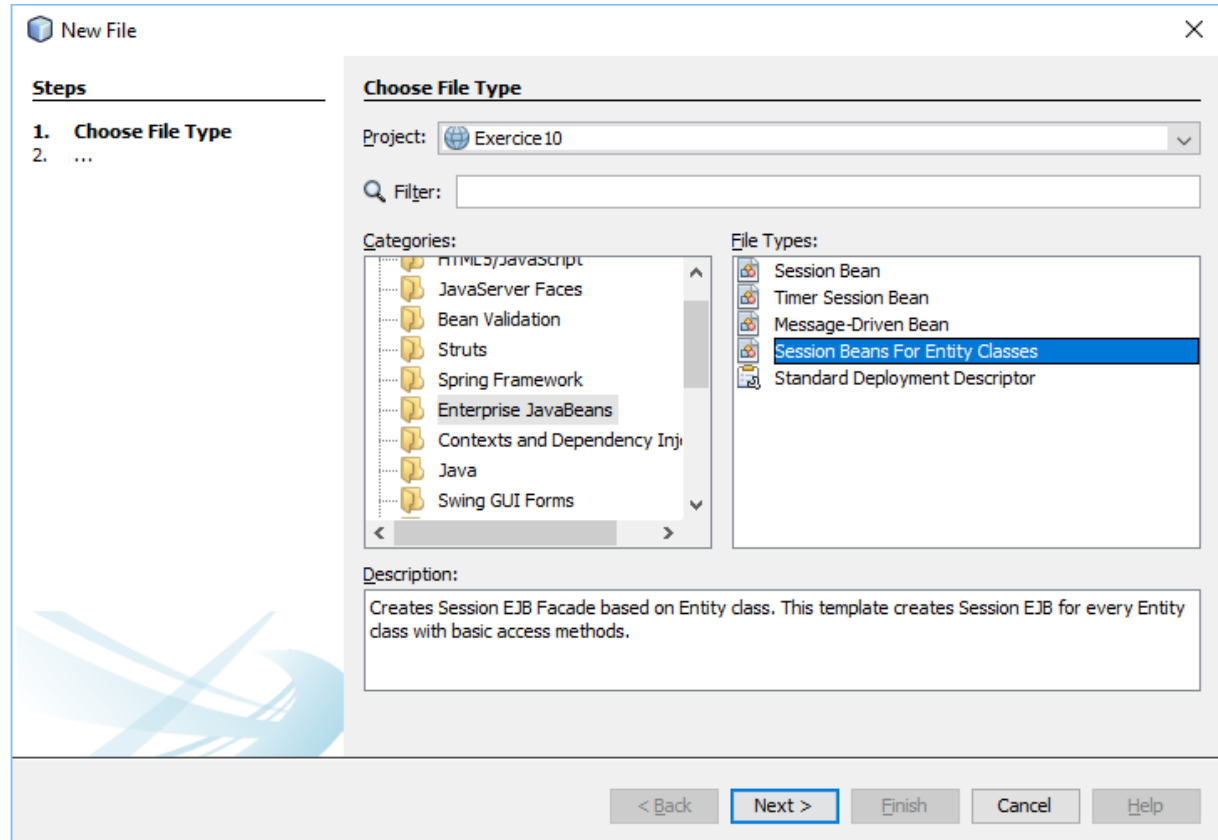
Exercice 10 : une façade pour les idées

Création d'un
EJB Session
façade vers
une entité
(étape 1)



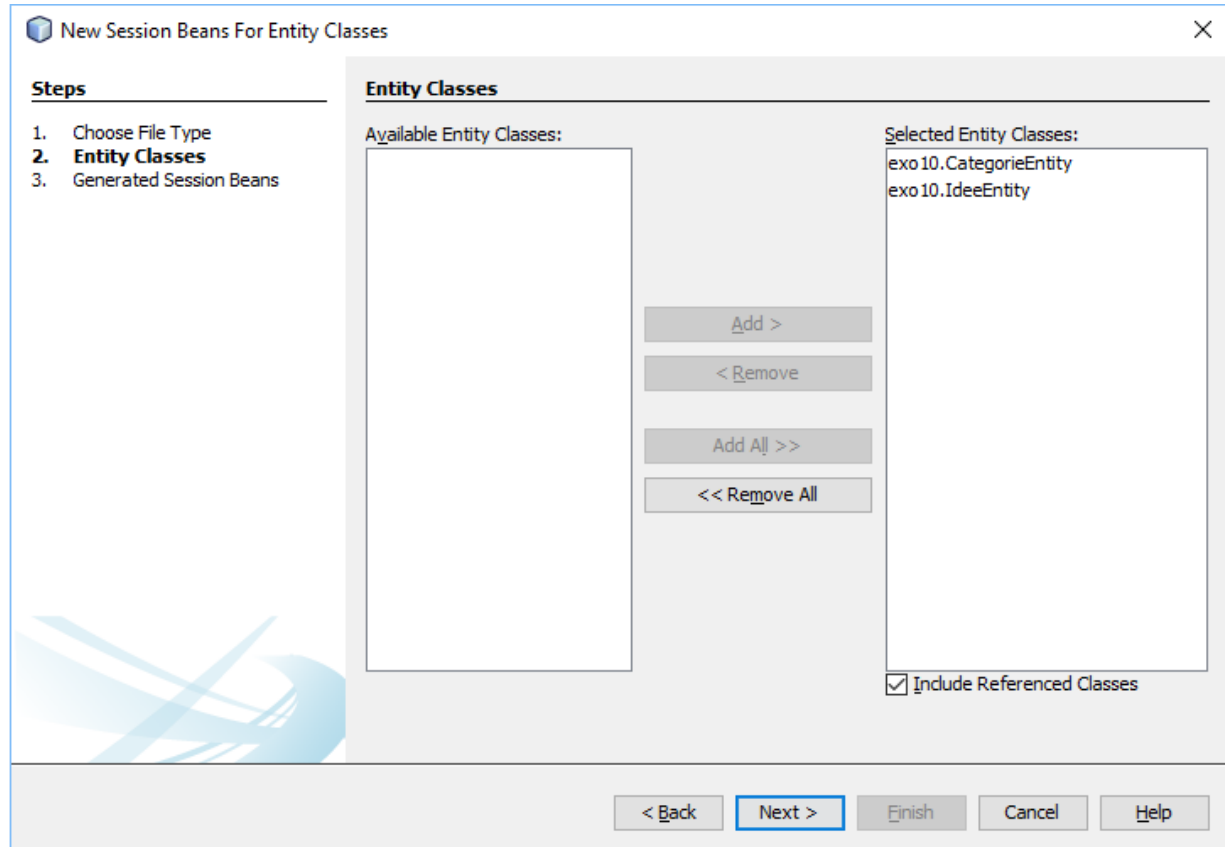
Exercice 10 : une façade pour les idées

Création d'un
EJB Session
façade vers
une entité
(étape 2)



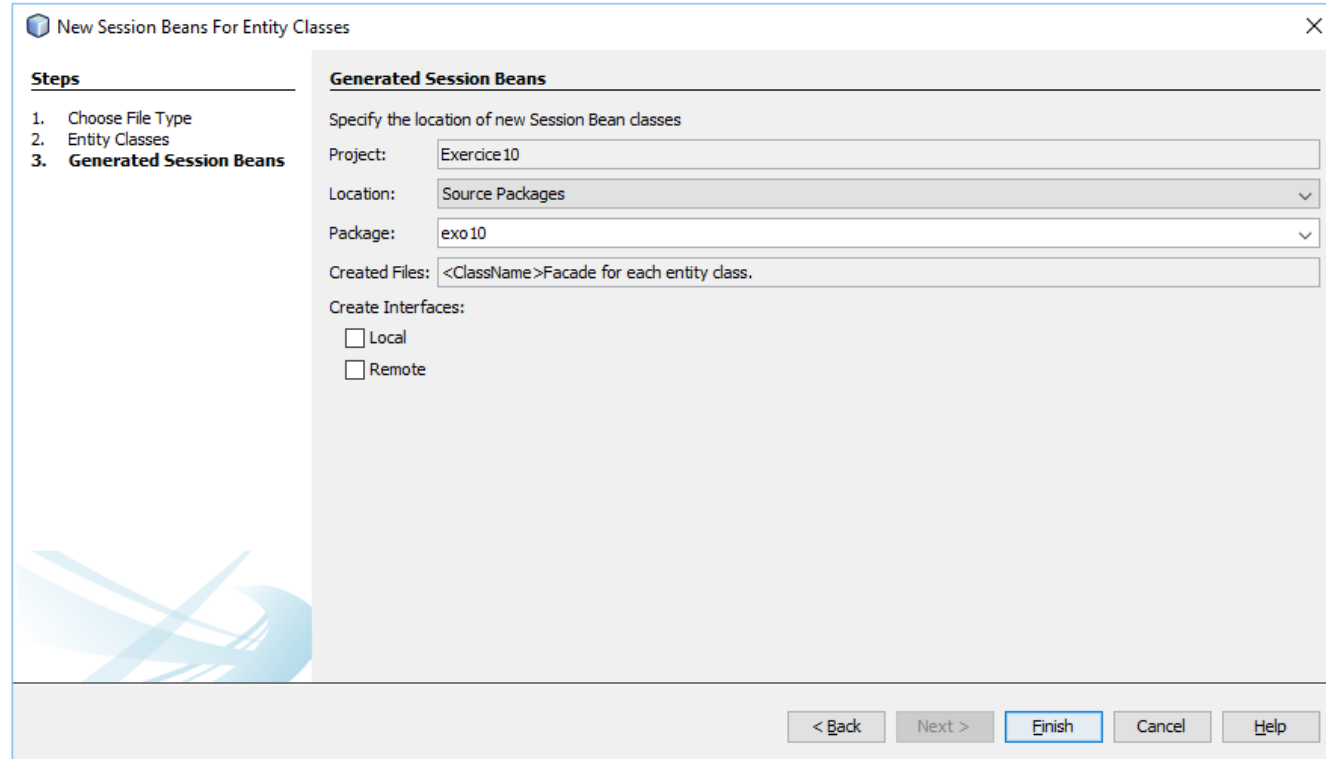
Exercice 10 : une façade pour les idées

Création d'un
EJB Session
façade vers
une entité
(étape 3)



Exercice 10 : une façade pour les idées

Création d'un
EJB Session
façade vers
une entité
(étape 4)



The screenshot shows the 'New Session Beans For Entity Classes' wizard in the Eclipse IDE. The wizard is titled 'New Session Beans For Entity Classes' and has a close button (X) in the top right corner. It is divided into two main sections: 'Steps' and 'Generated Session Beans'.

Steps:

1. Choose File Type
2. Entity Classes
3. **Generated Session Beans**

Generated Session Beans:

Specify the location of new Session Bean classes

Project:

Location:

Package:

Created Files:

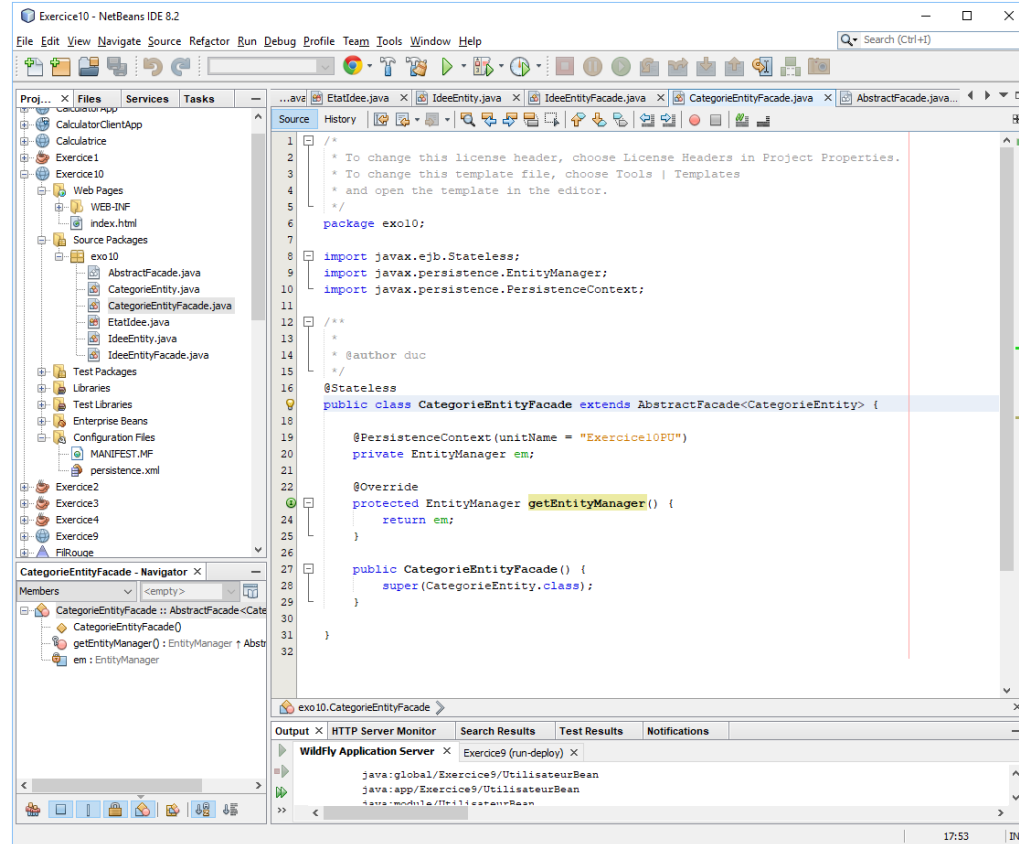
Create Interfaces:

☐ Local

☐ Remote

At the bottom of the wizard, there are five buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), 'Cancel', and 'Help'.

Exercice 10 : une façade pour les idées



Exercice 10 : une façade pour les idées

Ajouter les instructions suivantes dans la méthode annotée `@BeforeClass` de la classe de test – nécessaire pour obtenir un conteneur d'EJB embarqué :

```
Map<String, Object> properties = new HashMap();  
  
EJBContainer container =  
    javax.ejb.embeddable.EJBContainer.createEJBContainer  
        (properties);
```

Exercice 11 : Une communication simple par message

Exercice 11 : communication par message

On réalise un système de communication simple par message en point à point.

Une servlet minimaliste est chargée d'envoyer un message de type texte sur une queue JMS.

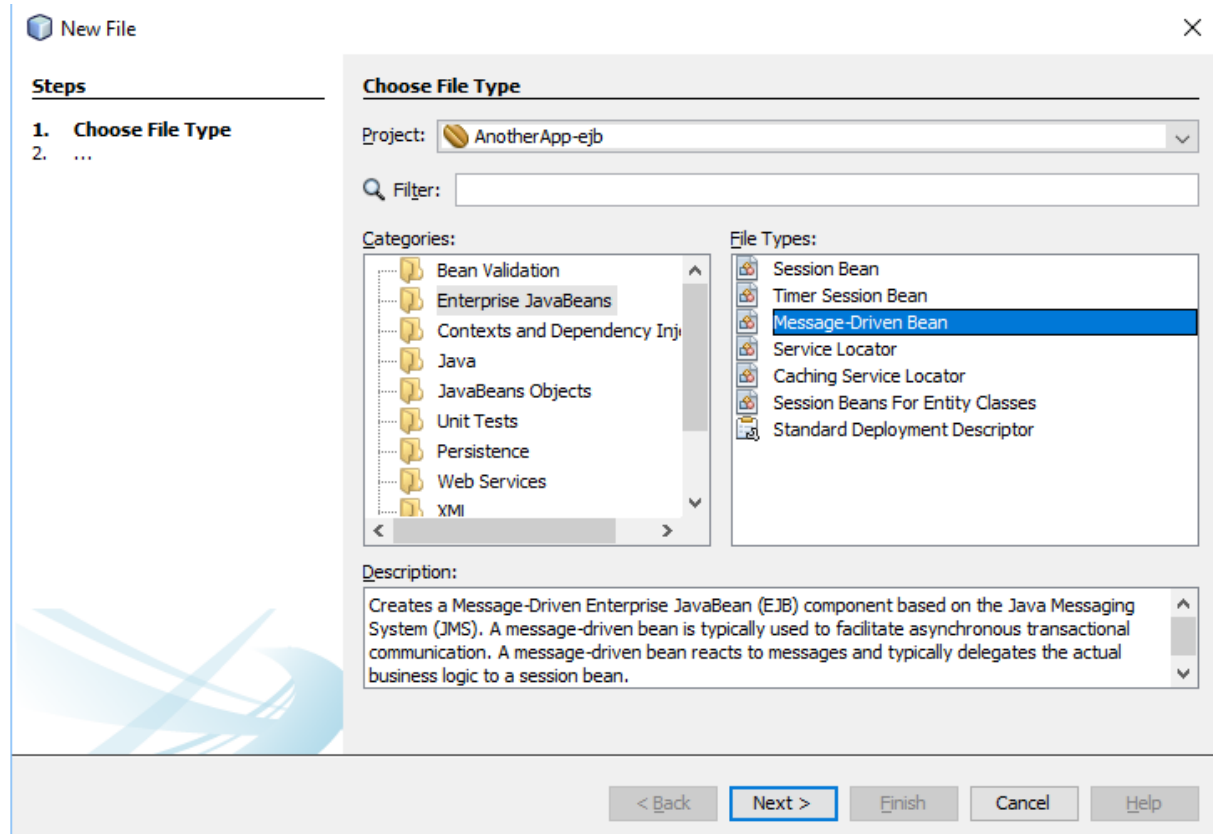
Un EJB “message-driven” est chargé de lire les messages et de les afficher.

On utilise GlassFish comme serveur d'application.

Le projet sera de type Java EE / Enterprise Application.

Exercice 11 : communication par message

Création d'un EJB
"message-driven"
- "étape 1"



Exercice 11 : communication par message

Création d'un EJB
"message-driven"
- "étape 2"

The screenshot shows the 'New Message-Driven Bean' wizard in the Eclipse IDE. The 'Steps' pane on the left indicates the current step is '2. Name and Location'. The main area shows the configuration for the new bean:

- Name and Location:**
 - EJB Name: NewMessageBean
 - Project: AnotherApp-ejb
 - Location: Source Packages
 - Package: ejb
- Destinations:**
 - ☒ Project Destinations: java:app/jms/NewMessage (with an 'Add...' button)
 - ☐ Server Destinations: jms/NewMessage

An 'Add Message Destination' dialog is open in the foreground, showing:

- Destination Name: FileDeMessages
- Destination Type: ☒ Queue (with ☐ Topic as an alternative)

Buttons at the bottom of the dialog include 'OK', 'Cancel', and 'Help'. The main wizard has '< Back', 'Next >', 'Finish', 'Cancel', and 'Help' buttons at the bottom.

Exercice 11 : communication par message

Création d'un EJB
"message-driven"
- "étape 3"

New File

Steps

1. Choose File Type
2. Name and Location
3. **Activation Config Properties**

Activation Config Properties

Property Name	Property Value
acknowledgeMode	AUTO_ACKNOWLEDGE
clientId	
connectionFactoryLookup	
destinationType	QUEUE
destinationLookup	java:app/FileDeMessages
messageSelector	
subscriptionDurability	NON_DURABLE
subscriptionName	

< Back Next > **Finish** Cancel Help

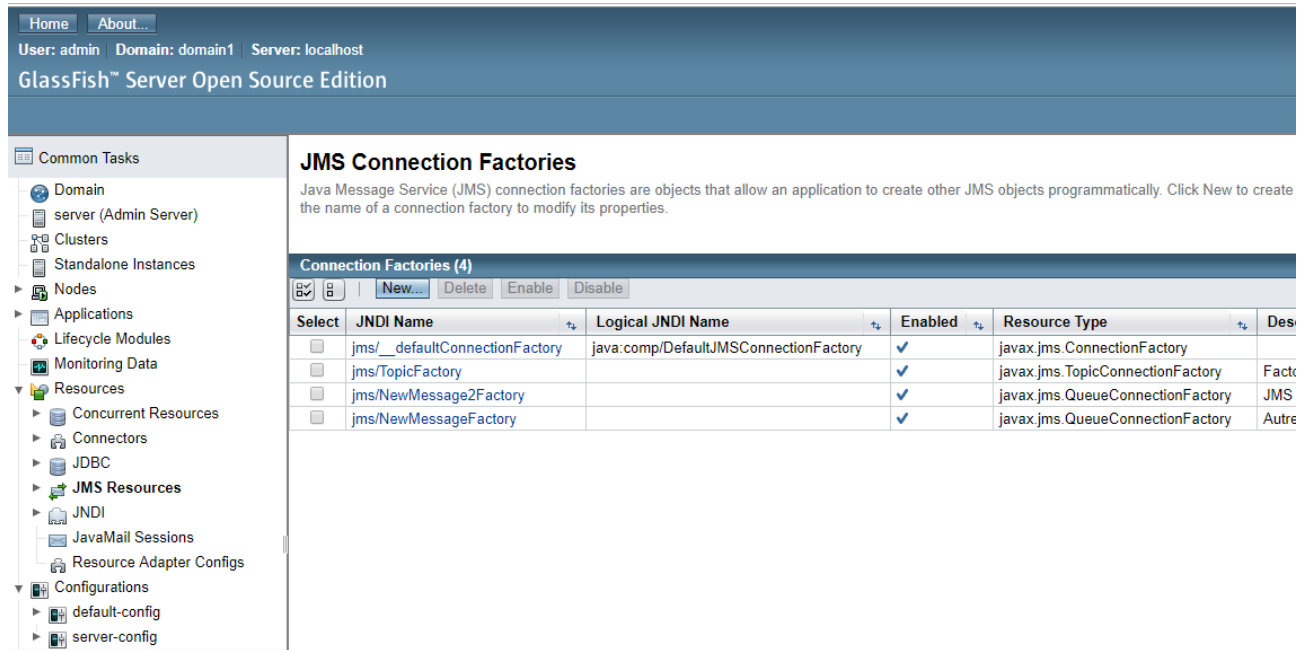
Exercice 11 : communication par message

Création d'une Connection Factory - étape 1



Exercice 11 : communication par message

Création d'une Connection Factory - étape 2



The screenshot shows the GlassFish Server Open Source Edition administration console. The top navigation bar includes 'Home' and 'About...' buttons, and displays 'User: admin | Domain: domain1 | Server: localhost'. The left sidebar shows a tree view of the server configuration, with 'JMS Resources' expanded under 'Resources'. The main content area is titled 'JMS Connection Factories' and includes a description: 'Java Message Service (JMS) connection factories are objects that allow an application to create other JMS objects programmatically. Click New to create the name of a connection factory to modify its properties.' Below this is a table titled 'Connection Factories (4)' with columns for 'Select', 'JNDI Name', 'Logical JNDI Name', 'Enabled', 'Resource Type', and 'Description'. The table lists four connection factories, all of which are enabled.

Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/_defaultConnectionFactory	java:comp/DefaultJMSConnectionFactory	✓	javax.jms.ConnectionFactory	
<input type="checkbox"/>	jms/TopicFactory		✓	javax.jms.TopicConnectionFactory	Fact
<input type="checkbox"/>	jms/NewMessage2Factory		✓	javax.jms.QueueConnectionFactory	JMS
<input type="checkbox"/>	jms/NewMessageFactory		✓	javax.jms.QueueConnectionFactory	Autre

Exercice 11 : communication par message

Création d'une Connection Factory - étape 3

Home About Help

User: admin Domain: domain1 Server: localhost
GlassFish™ Server Open Source Edition

Common Tasks

- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
- Configurations
 - default-config
 - server-config

New JMS Connection Factory

The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for the factory and a connector resource.

OK Cancel

General Settings

JNDI Name: *

Resource Type:

Description:

Status: ☒ Enabled

Pool Settings

Initial and Minimum Pool Size: Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: Connections
Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: Connections
Number of connections to be removed when pool idle timeout expires

Idle Timeout: Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: Milliseconds
Amount of time caller waits before connection timeout is sent

On Any Failure: ☐ Close All Connections
Close all connections and reconnect on failure, otherwise reconnect only when used

Transaction Support:
Level of transaction support. Overwrite the transaction support attribute in the Resource Adapter in a downward compatible way.

Connection Validation: ☐ Required
Validate connections, allow server to reconnect in case of failure

Additional Properties (0)

Add Property Delete Properties

Select	Name	Value	Description
No items found.			

Exercice 11 : communication par message

Création d'une Destination - étape 1

The screenshot shows the GlassFish Server Open Source Edition administration console. The top navigation bar includes 'Home' and 'About...' buttons, and displays the user 'admin', domain 'domain1', and server 'localhost'. The left sidebar shows a tree view of the server's configuration, with 'Resources' expanded to show 'JMS Resources'. The main content area is titled 'JMS Destination Resources' and contains a table of existing destination resources.

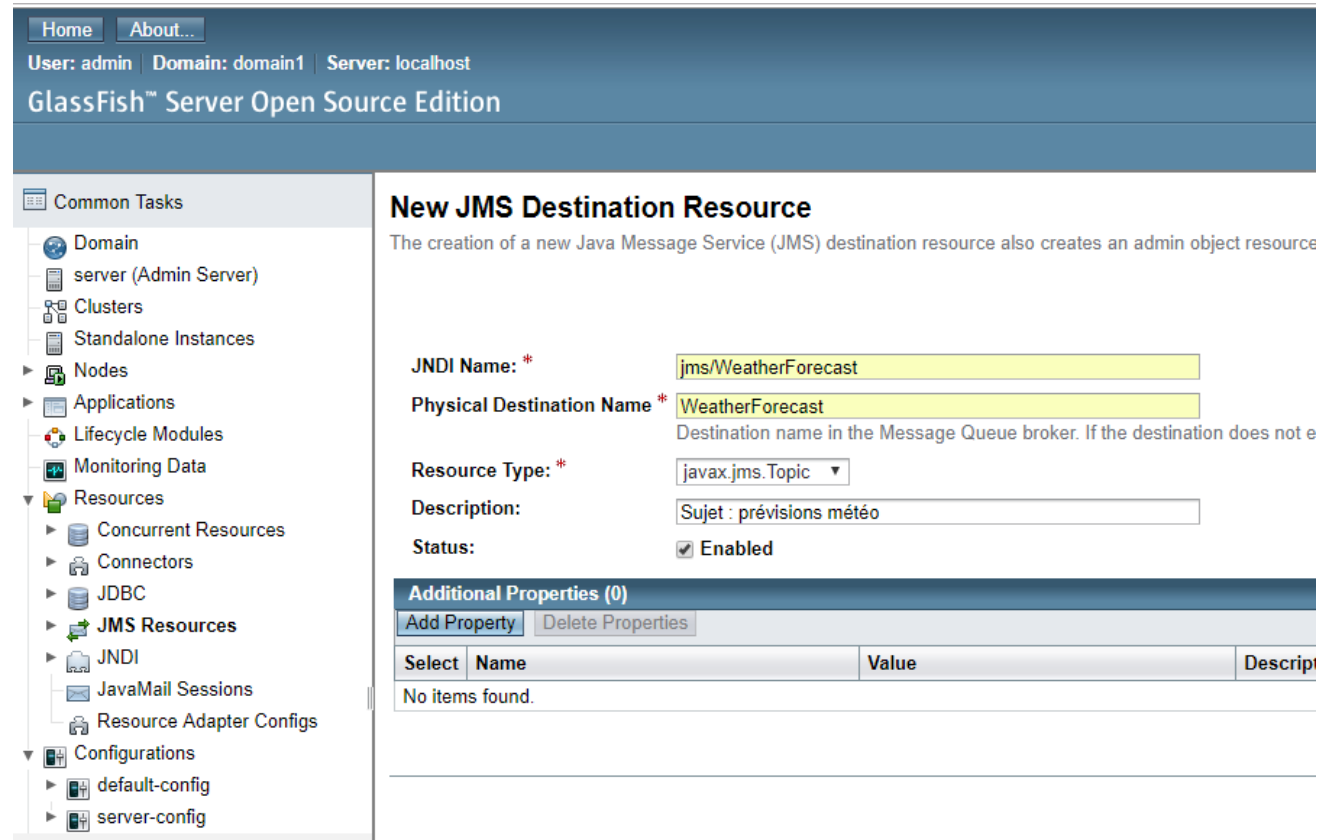
JMS Destination Resources

JMS destinations serve as the repositories for messages. Click New to create a new destination resource. Click the name of a destination resource to view its configuration.

Select	JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/NewMessage2	✓	javax.jms.Queue	
<input type="checkbox"/>	jms/NewMessage	✓	javax.jms.Queue	Queue po
<input type="checkbox"/>	jms/WeatherForecast	✓	javax.jms.Topic	Topic pou

Exercice 11 : communication par message

Création d'une Destination - étape 2



The screenshot shows the GlassFish Server Open Source Edition web console. The left sidebar contains a tree view of the server's configuration, with 'Resources' expanded to show 'JMS Resources'. The main panel is titled 'New JMS Destination Resource' and contains the following configuration fields:

- JNDI Name:** * jms/WeatherForecast
- Physical Destination Name:** * WeatherForecast
Destination name in the Message Queue broker. If the destination does not exist, it will be created.
- Resource Type:** * javax.jms.Topic
- Description:** Sujet : prévisions météo
- Status:** ☒ Enabled

Below the configuration fields is a section for 'Additional Properties (0)' with buttons for 'Add Property' and 'Delete Properties'. A table with columns 'Select', 'Name', 'Value', and 'Description' is shown, containing the text 'No items found.'

Exercice 12 : Une communication JMS avec objets

Exercice 12 : communication avec objets

On réalise un système un peu plus complexe de communication point à point par message contenant cette fois une entité JPA simple.

Une servlet toujours aussi minimaliste est chargée de créer une entité JPA et de l'envoyer sur une queue JMS.

Un EJB “message-driven” est chargé de lire le message, d'en extraire l'entité et de la persister dans une base de données précisée via une datasource existante.

On utilise GlassFish comme serveur d'application.

Le projet sera de type Java EE / Enterprise Application.

Exercice 13 : Une communication JMS en publish/subscribe

Exercice 13 : communication en pub/sub

Cette fois, on réalise un système de communication par message en mode publish/subscribe.

Une servlet minimaliste est chargée de publier un message texte sur un sujet (topic) quelconque.

Deux EJB “message-driven” s’abonnent aux messages sur ce sujet. Ils extraient le texte du message et l’affichent, chacun d’une manière un peu différente, pour qu’on puisse distinguer facilement qui a écrit quoi.

On utilise GlassFish comme serveur d’application.

Le projet sera de type Java EE / Enterprise Application.

Exercice 14 : Un service de courtage de valeurs boursières

Exercice 14 : accès à la Bourse

Une société de courtage en valeurs boursières décide de proposer un accès en ligne à la cotation de valeurs boursières :

- pour chaque société cotée, le système fournit la valeur de l'action de la société
- le système permet d'acquérir ou de vendre un certain nombre d'actions d'une société donnée.

Afin de permettre à ses clients d'accéder à ces fonctions au travers de leurs propres programmes, la société de courtage décide de les proposer sous forme d'un service web de type RPC.

Exercice 14 : accès à la Bourse

La valeur de l'action sera associée à un timestamp.

Le service d'achat / vente d'actions ne gèrera pas l'identification du client, mais retournera un identifiant unique de la transaction.

Les informations retournées (valeur d'action à un moment donné, opération d'achat ou de vente) seront enregistrées dans une base de données.

Le service de courtage fournira accès à un ensemble prédéfini d'actions dont la valeur pourra varier entre deux bornes fixes.

Exercice 14 : accès à la Bourse

NB important : cette liste d'actions ne devra être générée qu'une fois, pas lors de chaque appel du service de courtage.

Ce service sera testé par un client de type simple programme Java qui enregistrera les valeurs d'action et opérations.

Il est conseillé de définir complètement le service de courtage avant de développer le client...

Côté service de courtage, on choisira un projet de type Java Web Application et GlassFish comme serveur d'applications.

Exercice 15 : Une bibliothèque électronique

Exercice 15 : bibliothèque électronique

Une importante bibliothèque souhaite permettre l'accès à ses services par programme.

Ses services consistent en :

- la création, la consultation, la modification et la suppression d'utilisateurs
- l'ajout, la consultation, la modification et le retrait de livres dans la bibliothèque.

Un utilisateur est décrit par ses nom, prénom et adresse mail.

Un livre contient un titre, un auteur, une catégorie et un contenu.

Exercice 15 : bibliothèque électronique

Il est demandé de pouvoir obtenir la représentation des utilisateurs et livres en XML ou en texte simple.

Pour la création d'un utilisateur, il est convenu que c'est la bibliothèque électronique qui génère son identifiant.

Pour l'ajout d'un livre, c'est au contraire le programme client qui choisira son identifiant.

MERCI !

& SUIVEZ-NOUS !



HUMAN**booster**
VOIRE SOLUTION COMPÉTENCE

04 73 24 93 11

contact@humanbooster.com

www.humanbooster.com

@ Patrick Duc 2018