Support Formation JAVA EJB3

Pour	Formation
Date	08/11/2024
Objet	Support Formation JAVA EJB3

I) Vocabulaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.

- **ORM** : Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

- **Hibernate** : Hibernate est un framework open source gérant la persistance des objets en base de données relationnelle. Hibernate est adaptable en termes d'architecture, il peut donc être utilisé aussi bien dans un développement client lourd, que dans un environnement web léger de type **Apache Tomcat** ou dans un environnement Java EE complet : **WebSphere, JBoss Application Server et Oracle WebLogic Server**. Hibernate apporte une solution aux problèmes d'adaptation entre le paradigme objet et les SGBD en remplaçant les accès à la base de données par des appels à des méthodes objet de haut niveau.

- **TopLink** : TopLink est un framework de mapping objet-relationnel pour le développement Java. Il fournit une plateforme puissante et flexible permettant de stocker des objets Java dans une base de données relationnelle et/ou de les convertir en documents XML. TopLink Essentials est la version open source du produit d'Oracle.

- **EclipseLink** : EclipseLink est un framework open source de mapping objet-relationnel pour les développeurs Java. Il fournit une plateforme puissante et flexible permettant de stocker des objets Java dans une base de données relationnelle et/ou de les convertir en documents XML. EclipseLink est dérivé du projet **TopLink** de la société **Oracle**. Il supporte un certain nombre d'API relatives à la persistance des données et notamment la JPA.

- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.

- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le en HQL le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.

- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.

- **EJB** : **Enterprise Java Beans** (EJB) est une architecture de composants logiciels côté serveur pour la plateforme de développement Java EE. Cette architecture propose un cadre pour créer des composants distribués (c'est-à-dire déployés sur des serveurs distants) écrit en langage de programmation Java hébergés au sein d'un serveur applicatif permettant de représenter des données (EJB dit entité), de proposer des services avec ou sans conservation d'état entre les appels (EJB dit session), ou encore d'accomplir des tâches de manière asynchrone (EJB dit message). Tous les EJB peuvent évoluer dans un contexte transactionnel ce qui peut permettre de gérer les transactions avec les sources de données (fichier Xml, fichier Csv, fichier Json, base de donnée etc.…).

- **POJO** : POJO est un acronyme qui signifie Plain Old Java Object que l'on peut traduire en français par bon vieil objet Java. Cet acronyme est principalement utilisé pour faire référence à la simplicité d'utilisation d'un objet Java en comparaison avec la lourdeur d'utilisation d'un composant EJB. Ainsi, un POJO n'implémente pas d'interface spécifique à un Framework comme c'est le cas par exemple pour un composant EJB.

- **POJI** : POJI est un acronyme qui signifie Plain Old Java Interfaces que l'on peut traduire en français par bon vieil Interface Java correspond à une interface standard Java. Ils sont habituellement utilisés dans le contexte JEE pour fournir des services.

- **Servlet** : Une servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Ces données sont le plus généralement présentées au format HTML, mais elles peuvent également l'être au format XML ou tout autre format destiné aux navigateurs web. Les servlets utilisent l'API Java Servlet (package **javax.servlet**). Une servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier, typiquement : accès à des bases de données, transactions d'e-commerce, etc. Une servlet peut être chargé automatiquement lors du démarrage du serveur web ou lors de la première requête du client, une fois chargés, les servlets restent actifs dans l'attente d'autres requêtes du client.

- **Filtre de servlet** : Un filtre HTTP de servlet est un composant d'une application web qui agit comme un intercepteur sur une servlet. Un filtre est une classe Java qui implémente l'interface javax.servlet.Filter. Il est déclaré dans le descripteur de l'application web.xml, et posé sur une ou plusieurs servlets. Lorsqu'une requête HTTP doit être traitée par une servlet sur laquelle un filtre est appliqué, alors le serveur va exécuter la méthode **doFilter** du Filtre avant d'exécuter la méthode **doGet** ou **doPost** de la servlet.

- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. Selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.

- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.

- **JNDI** : JNDI signifie Java Naming and Directory Interface, cette API permet d'accéder à différents services de nommage ou de répertoire de façon uniforme, d'organiser et rechercher des informations ou des objets par nommage (java naming and directory interface), de faire des opérations sur des annuaires (java naming and directory interface) tels que : LDAP : un annuaire léger, X500 : normes d'annuaires lourdes à mettre en œuvre, NIS : annuaire obsolète.

- **Pool de connexions** : Un pool de connexions est un mécanisme permettant de réutiliser les connexions créées. En effet, la création systématique de nouvelles instances de Connection peut parfois devenir très lourd en consommation de ressources. Pour éviter cela, un pool de connexions ne ferme pas les connexions lors de l'appel à la méthode close(). Au lieu de fermer directement la connexion, celle-ci est « retournée » au pool et peut être utilisée ultérieurement.La gestion du pool se fait en général de manière transparente pour l'utilisateur.

II) Design pattern Injection de dépendance.

Considérons une entreprise avec des employés dont chacun a une adresse et un rôle précis dans l'entreprise. Les entités mis en jeu dans cet example son **Employe**, **Role** et **Adresse** dont les signatures sont :

```
1   public class Role {
2       private int idRole;
3       private String codeRole;
4       private String descriptionRole;
5       public Role() {}
6   }
7   public class Adresse {
8       private int idAdresse;
9       private String nomRue;
10      private String codePostal;
11      private String ville;
12      private String pays;
13      public Adresse() {}
14  }
15  public class Employe {
16      private int idEmploye;
17      private Role role;
18      private Adresse adresse;
19      public Employe() {
20          this.role = new Role();
21          this.adresse = new Adresse();
22      }
23  }
```

En version copiable :

```
public class Role {
   private int idRole;
   private String codeRole;
   private String descriptionRole;
   public Role() {}
}
public class Adresse {
   private int idAdresse;
   private String nomRue;
   private String codePostal;
   private String ville;
   private String pays;
   public Adresse() {}
}
public class Employe {
   private int idEmploye;
   private Role role;
   private Adresse adresse;
   public Employe() {
```

```
    this.role = new Role();
    this.adresse = new Adresse();
  }
}
```

Ce code ci-dessus permet de voir la relation de dépendance entre la classe **Employe** et les classes **Role** et **Adresse**. Mais le code comporte quand même quelques soucis :

- Lorsqu'on instancie un objet de type **Employe** alors les objets **Role et Adresse** sont tout temps les mêmes.
- Lorsqu'on change la signature des classes **Role** ou **Adresse** (constructeur par exemple) alors on doit aussi changer le constructeur de la classe **Employe** aussi sous peine d'avoir des erreurs de compilation.

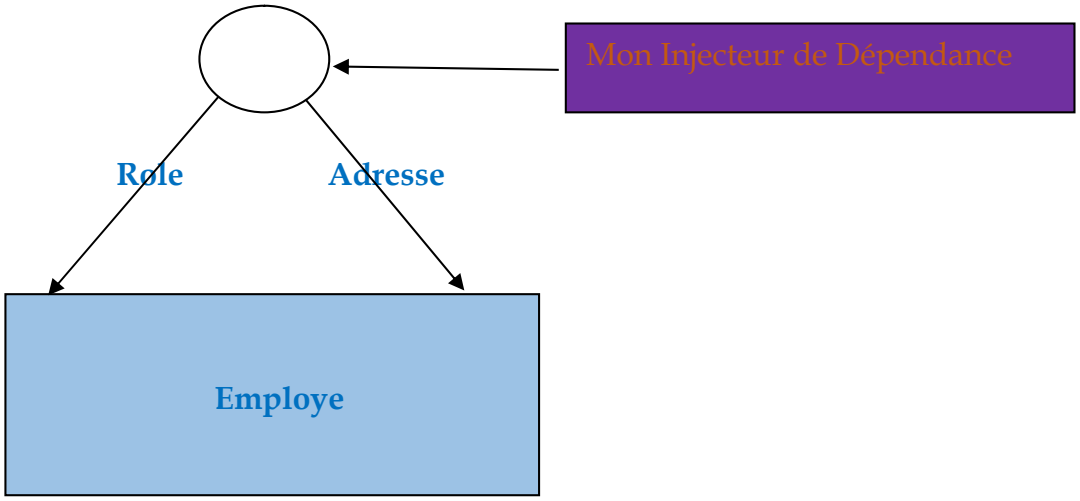
Le premier problème peut être résolu facilement en redéfinissant les constructeurs **Role** et **Adresse**. Ceci va nous obliger à changer les classes **Employe**, **Role** et **Adresse**.

```
public class Role {
    private int idRole;
    private String codeRole;
    private String descriptionRole;

    public Role(String codeRole, String descriptionRole) {
        this.codeRole = codeRole;
        this.descriptionRole = descriptionRole;
    }
}

public class Adresse {
    private int idAdresse;
    private String nomRue;
    private String codePostal;
    private String ville;
    private String pays;

    public Adresse(String nomRue, String codePostal, String ville, String pays) {
        this.nomRue = nomRue;
        this.codePostal = codePostal;
        this.ville = ville;
        this.pays = pays;
    }
}

public class Employe {
    private int idEmploye;
    private Role role;
    private Adresse adresse;

    public Employe(Role role,Adresse adresse) {
        this.role = role;
        this.adresse = adresse;
    }
}
Role role = new Role("Administrateur","Je suis Admin");
Address addresse = new Address("river","5555","New York","Etats Unis");
Employe employe = new Employe(role,addresse);

```

En version copiable :

```
public class Role {
    private int idRole;
    private String codeRole;
    private String descriptionRole;

    public Role(String codeRole, String descriptionRole) {
        this.codeRole = codeRole;
        this.descriptionRole = descriptionRole;
    }
}

public class Adresse {
    private int idAdresse;
    private String nomRue;
    private String codePostal;
    private String ville;
    private String pays;

    public Adresse(String nomRue, String codePostal, String ville, String pays) {
        this.nomRue = nomRue;
        this.codePostal = codePostal;
        this.ville = ville;
        this.pays = pays;
    }
}

public class Employe {
    private int idEmploye;
    private Role role;
    private Adresse adresse;

    public Employe(Role role,Adresse adresse) {
        this.role = role;
        this.adresse = adresse;
    }
}
Role role = new Role("Administrateur","Je suis Admin");
Address addresse = new Address("river","5555","New York","Etats Unis");
Employe employe = new Employe(role,addresse);
```

Lignes 37 ,38, 39 : Pour avoir une instance de **Employe** il faut aussi avoir une instance de **Role** et **Adresse** ce qui cause aussi problème.
Ce qui serait trop cool c'est de pouvoir avoir une instance de **Employe** sans se souci des instances qui servent à construire un objet de type **Employe** (ici **Role** et **Adresse**).



Dans l'exemple ci dessus c'est le framework « **Mon injecteur de dépendance** » qui se charge d'injecter les instances necessaire à la contruction d'un objet de type **Employe** permettant ainsi de renvoyer des instances d'**Employe** sans se soucier de **Role** et **Adresse**.
En POO (programmation orienté objet) c'est ce processus que l'on appelle Injection de dépendance. Dans les langages tel que **Java** et **.Net** ce processus d'injection est géré par un Framework portant le nom de **Spring**.

III) Généralités sur les EJB

Enterprise Java Beans (EJB) est une architecture de composants logiciels côté serveur pour la plateforme de développement Java EE. Cette architecture propose un cadre pour créer des composants distribués (c'est-à-dire déployés sur des serveurs distants) écrit en langage de programmation Java hébergés au sein d'un serveur applicatif permettant de représenter des données (EJB dit entité), de proposer des services avec ou sans conservation d'état entre les appels (EJB dit session), ou encore d'accomplir des tâches de manière asynchrone (EJB dit message). Tous les EJB peuvent évoluer dans un contexte transactionnel ce qui peut permettre de gérer les transactions avec les sources de données (fichier Xml, fichier Csv, fichier Json, base de données etc.…).
Les accès aux EJB par un client se font obligatoirement par le conteneur à EJB.
Un conteneur d'EJB propose un certain nombre de services qui assurent la gestion :

- Du cycle de vie du Java Bean
- De l'accès au Java Bean
- De la sécurité d'accès
- Des accès concurrents
- Des transactions

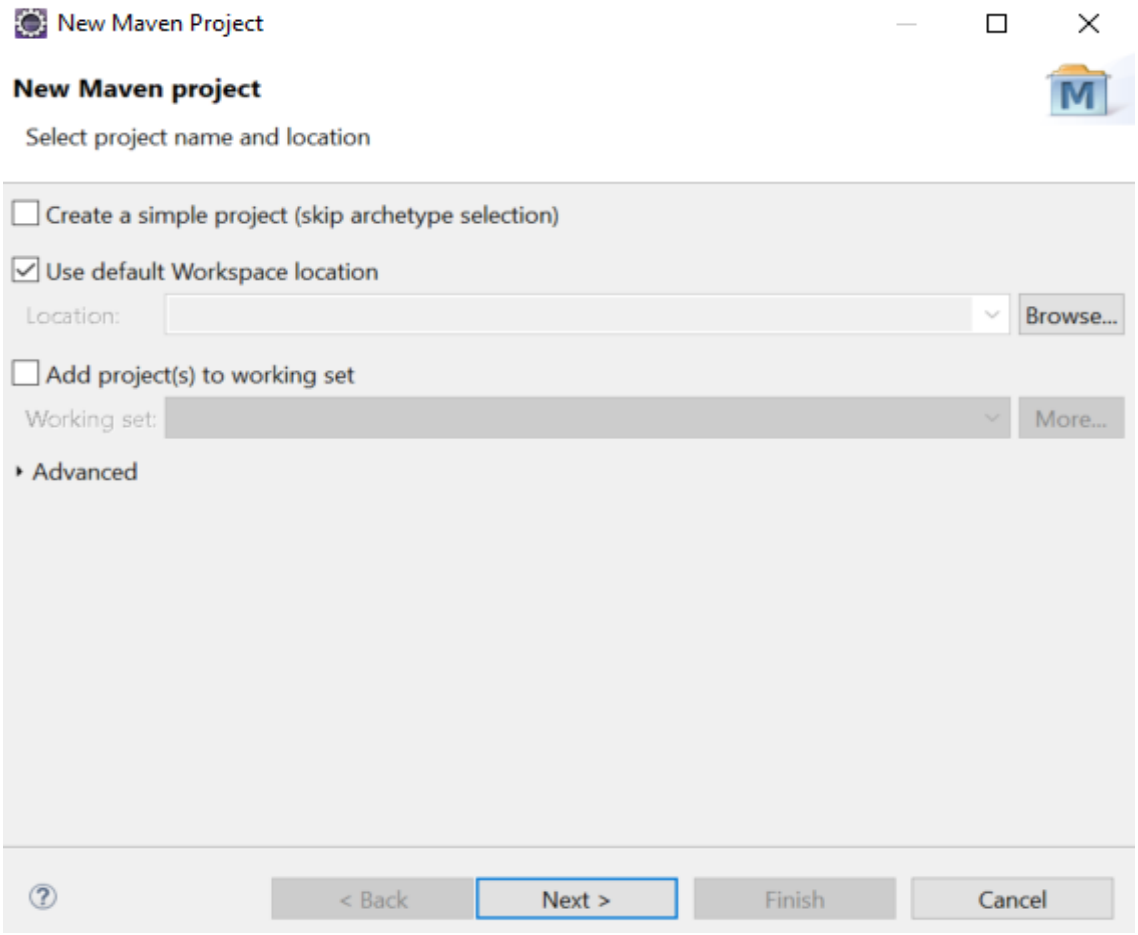
Il existe trois types d'EJB :

- Les Beans entité (Entity Beans).
- Les Beans session : Session Beans de type **Stateless** en JEE5 et type **Singleton** en JEE6 (ne conserve pas d'état du Bean entre les différents appels) ou de type **Stateful** (conserve l'état du Bean entre les différents appels).
- Les Beans message (message Driven Beans).

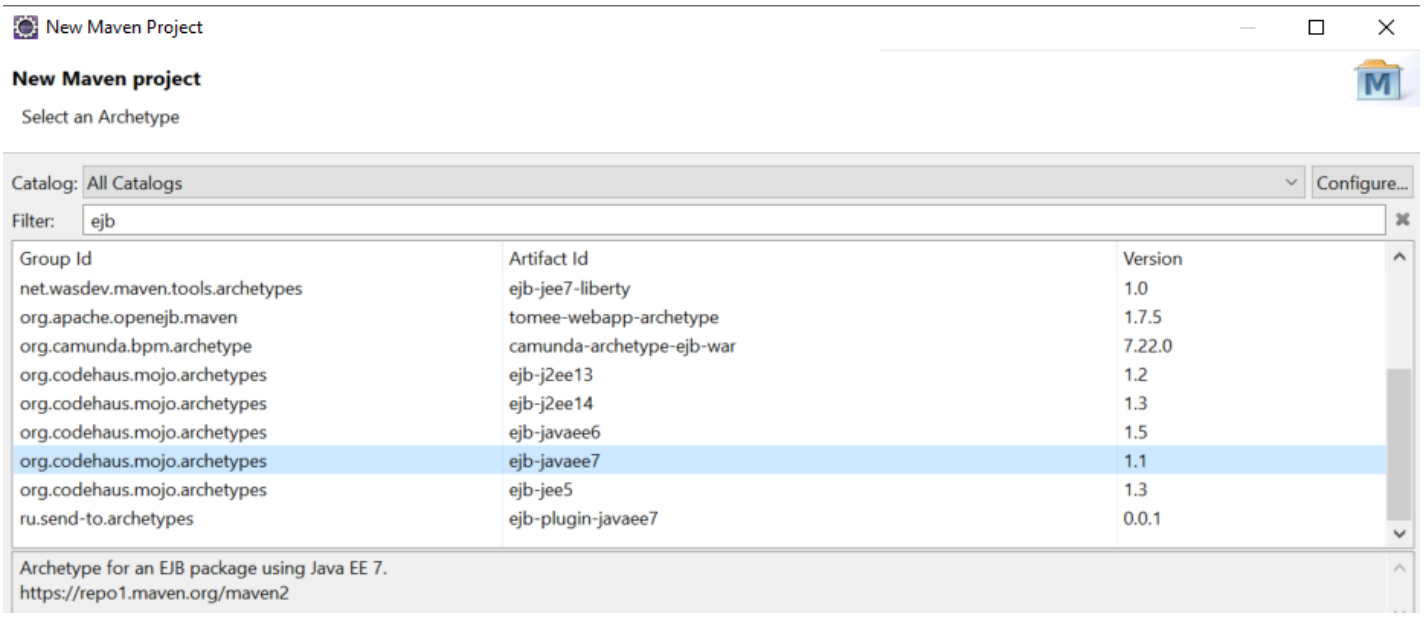
IV) Création d'un serveur et client EJB

1. Serveur EJB

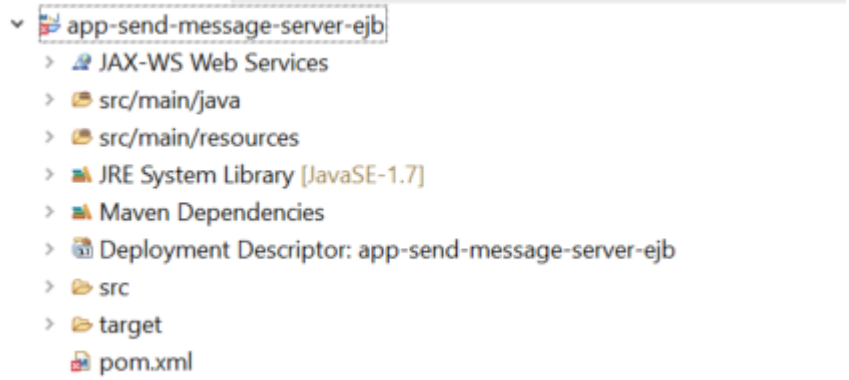
Créer le projet **app-send-message-server-ejb** qui sera notre Serveur EJB.

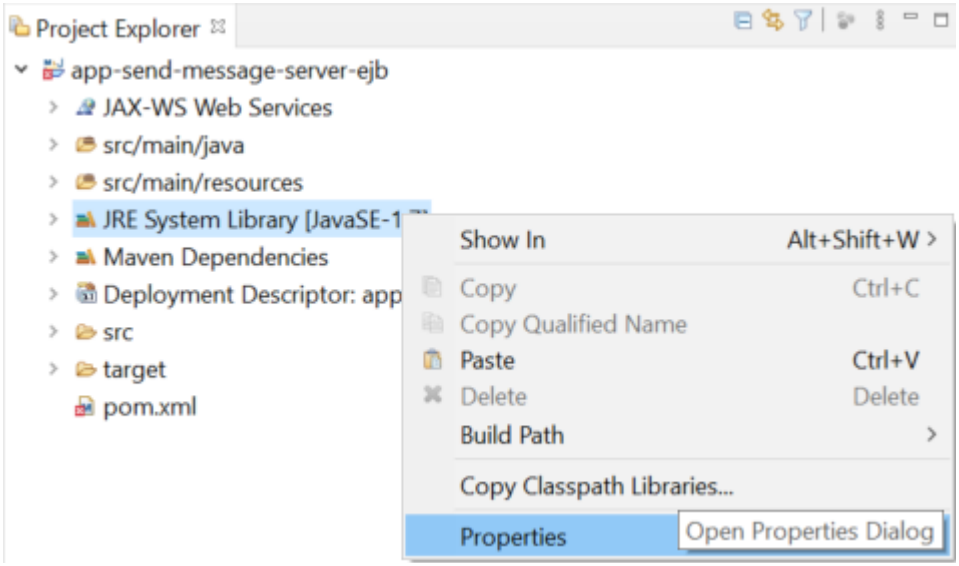


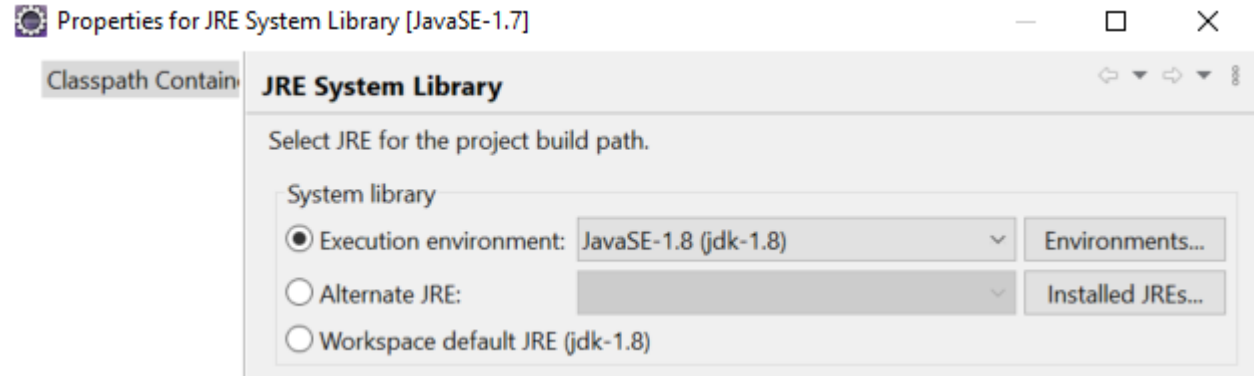
Taper sur Filter **ejb**.




New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.training
Artifact Id: app-send-message-server-ejb
Version: 0.0.1-SNAPSHOT
Package: com.training

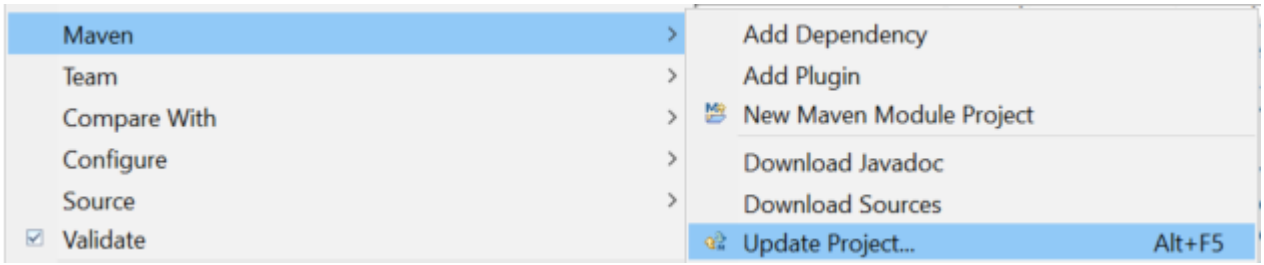

app-send-message-server-ejb
JAX-WS Web Services
src/main/java
src/main/resources
JRE System Library [JavaSE-1.7]
Maven Dependencies
Deployment Descriptor: app-send-message-server-ejb
src
target
pom.xml

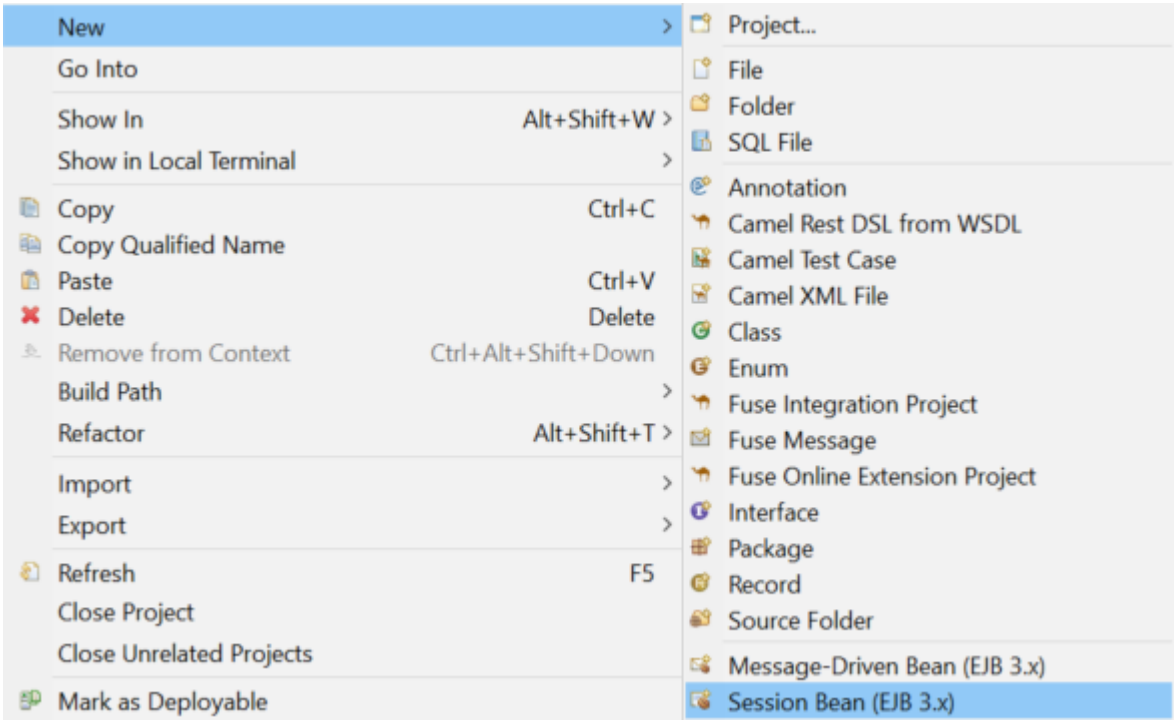

Project Explorer
app-send-message-server-ejb
JAX-WS Web Services
src/main/java
src/main/resources
JRE System Library [JavaSE-1
Maven Dependencies
Deployment Descriptor: app
src
target
pom.xml
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Copy Classpath Libraries...
Properties
Open Properties Dialog

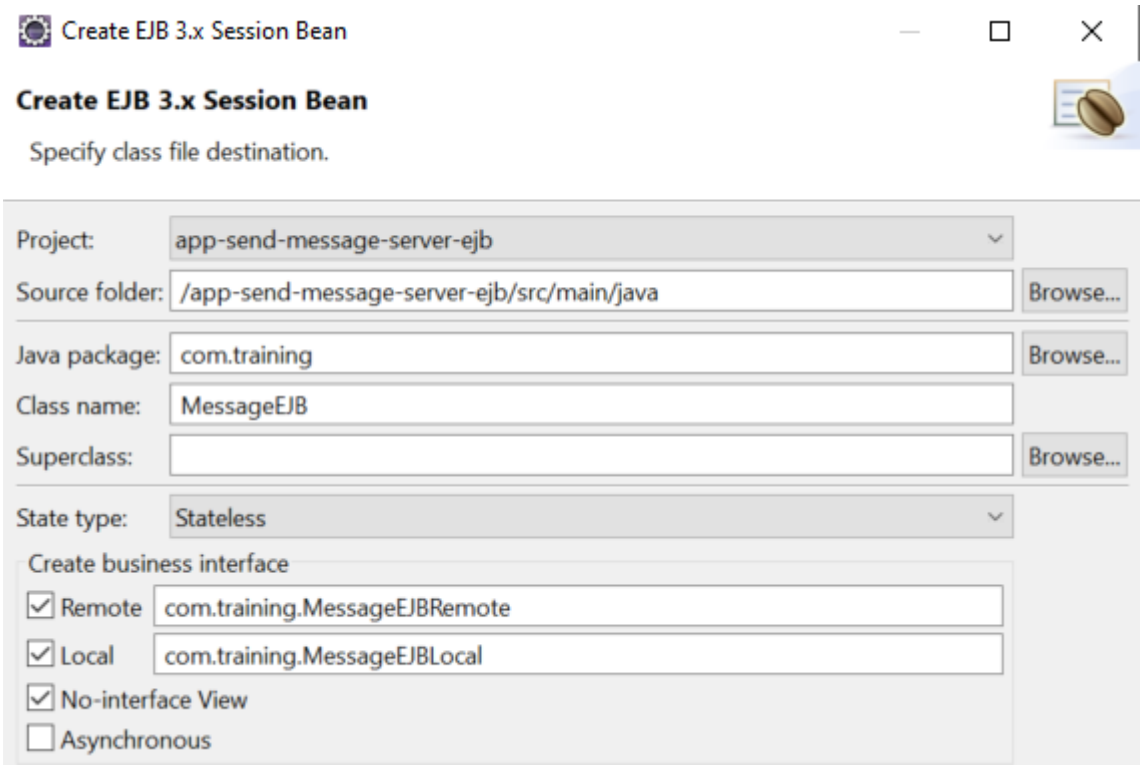

Properties for JRE System Library [JavaSE-1.7]
Classpath Contain
JRE System Library
Select JRE for the project build path.
System library
Execution environment: JavaSE-1.8 (jdk-1.8)
Environments...
Alternate JRE:
Installed JREs...
Workspace default JRE (jdk-1.8)

Ajouter la balise **<pluginManagement>** avant la balise **<plugins>**.

Créer les dossier **src/test/main** et **src/test/resources** s'il n'existe pas.







On obtient les classes ci-dessous :
La classe **com.training.MessageEJB** aura comme contenu :

```
package com.training;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

/**
 * Session Bean implementation class MessageEJB
 */
@Stateless
@LocalBean
public class MessageEJB implements MessageEJBRemote, MessageEJBLocal {

    /**
     * Default constructor.
     */
    public MessageEJB() {

    }

    @Override
    public String sendMessage() {
        return "this is our EJB Message";
    }
}
```

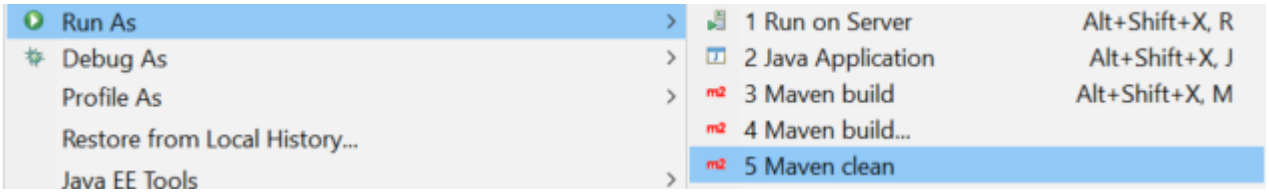
La classe **com.training.MessageEJBLocal** aura comme contenu :

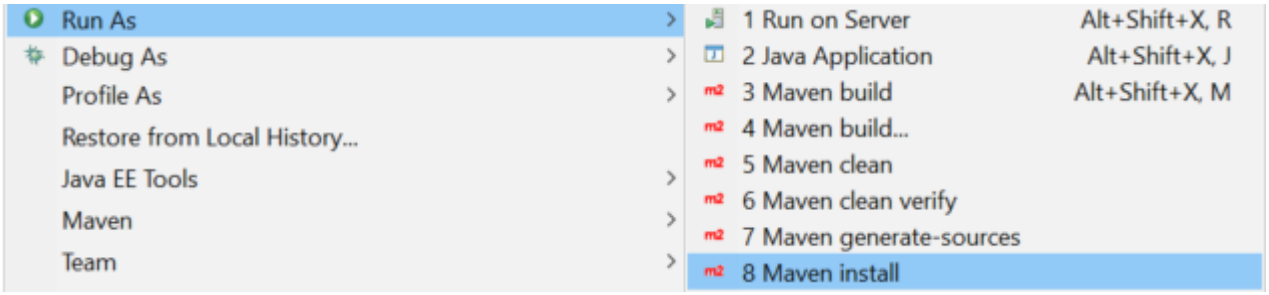
```
package com.training;

import javax.ejb.Local;

@Local
public interface MessageEJBLocal {
   public String sendMessage();
}
```

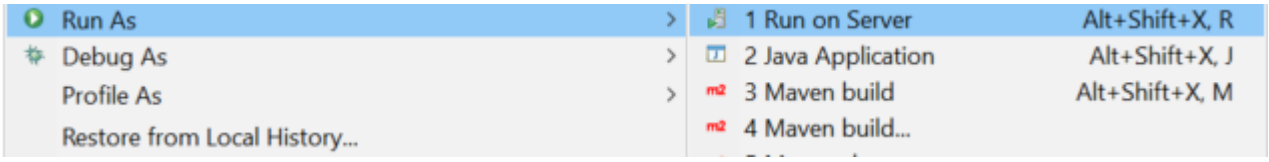
La classe **com.training.MessageEJBRemote** aura comme contenu :

```
package com.training;

import javax.ejb.Remote;

@Remote
public interface MessageEJBRemote {

   public String sendMessage();
}
```

Lancer un maven clean et install du projet **app-send-message-server-ejb** et deployer le code sur le serveur **Wildfly**.



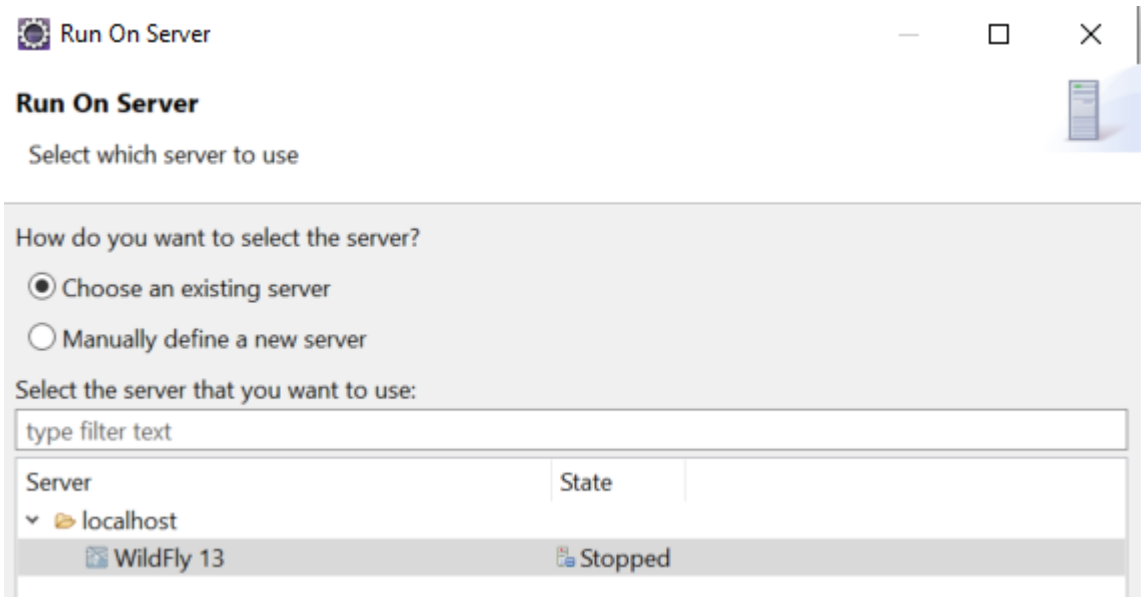


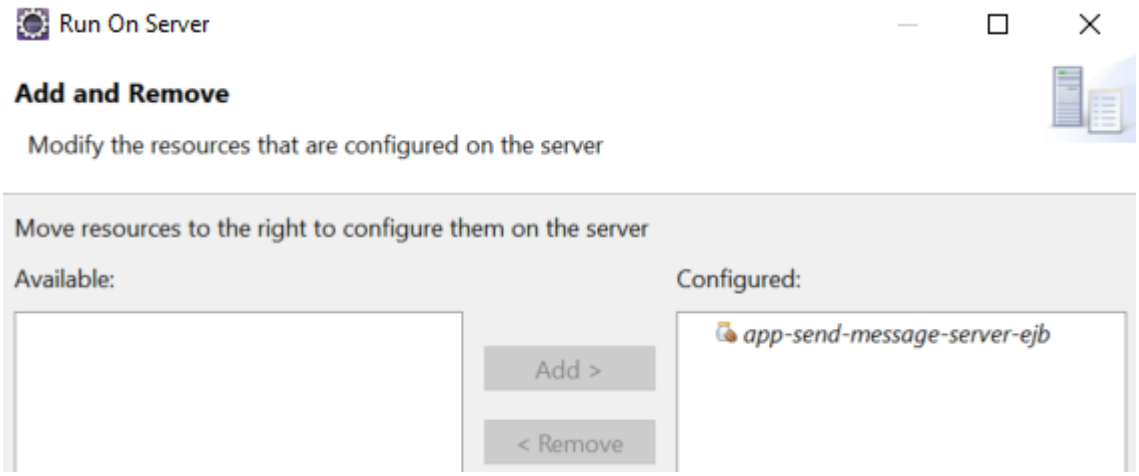


Le fichier **pom.xml** devient :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.training</groupId>
  <artifactId>app-send-message-server-ejb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>ejb</packaging>

  <name>app-send-message-server-ejb</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.1</version>
          <configuration>
            <source>${maven.compiler.source}</source>
            <target>${maven.compiler.target}</target>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-ejb-plugin</artifactId>
          <version>2.3</version>
          <configuration>
            <ejbVersion>3.1</ejbVersion>
          </configuration>
```

```
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-dependency-plugin</artifactId>
          <version>2.6</version>
          <executions>
            <execution>
              <phase>validate</phase>
              <goals>
                <goal>copy</goal>
              </goals>
              <configuration>
                <outputDirectory>${endorsed.dir}</outputDirectory>
                <silent>true</silent>
                <artifactItems>
                  <artifactItem>
                    <groupId>javax</groupId>
                    <artifactId>javaee-endorsed-api</artifactId>
                    <version>7.0</version>
                    <type>jar</type>
                  </artifactItem>
                </artifactItems>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>

</project>
```





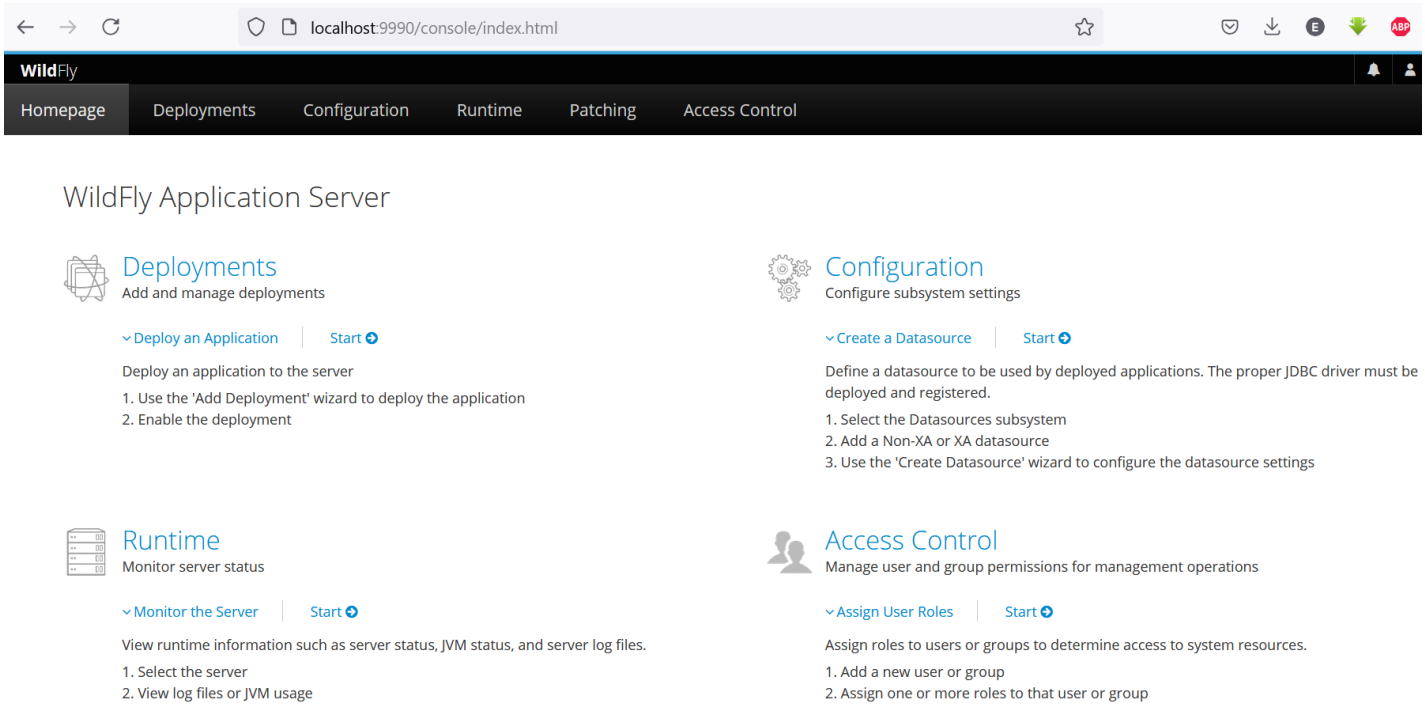
On voit sur la console :

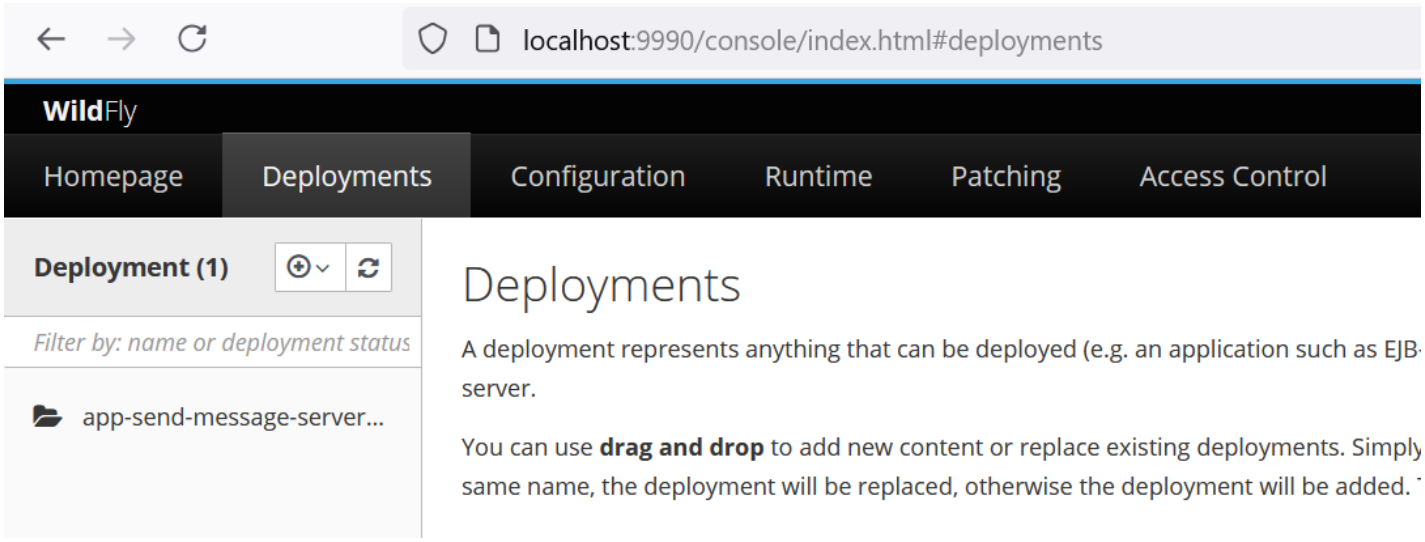
```
20:23:18,882 INFO  [org.wildfly.extension.undertow] (MSC service thread 1-2) WFLYUT0006: Undertow HTTPS liste
20:23:19,525 INFO  [org.jboss.ws.common.management] (MSC service thread 1-1) JBWS022052: Starting JBossWS 5.2
20:23:19,941 INFO  [org.jboss.weld.deployer] (MSC service thread 1-2) WFLYWELD0003: Processing weld deploymen
20:23:20,098 INFO  [org.hibernate.validator.internal.util.Version] (MSC service thread 1-2) HV000001: Hiberna
20:23:20,144 INFO  [org.jboss.as.ejb3.deployment] (MSC service thread 1-2) WFLYEJB0473: JNDI bindings for ses

        java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
        java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
        java:module/MessageEJB!com.training.MessageEJB
        ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
        java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
        java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
        java:module/MessageEJB!com.training.MessageEJBLocal
        ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
        java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
        java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
        java:module/MessageEJB!com.training.MessageEJBRemote
        java:jboss/exported/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
        ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote

20:23:20,253 INFO  [org.jboss.weld.Version] (MSC service thread 1-5) WELD-000900: 3.0.4 (Final)
20:23:20,471 INFO  [org.infinispan.factories.GlobalComponentRegistry] (MSC service thread 1-4) ISPN000128: In
20:23:20,628 INFO  [org.jboss.as.clustering.infinispan] (ServerService Thread Pool -- 66) WFLYCLINF0002: Star
20:23:20,702 WARN  [org.jboss.weld.Bootstrap] (MSC service thread 1-6) WELD-000146: BeforeBeanDiscovery.addAn
20:23:20,715 WARN  [org.jboss.weld.Bootstrap] (MSC service thread 1-6) WELD-000146: BeforeBeanDiscovery.addAn
```

On obtient au niveau du JNDI bindings la liste ci-dessous :

java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
java:module/MessageEJB!com.training.MessageEJB
ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJB
java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
java:module/MessageEJB!com.training.MessageEJBLocal
ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJBLocal
java:global/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
java:app/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
java:module/MessageEJB!com.training.MessageEJBRemote
java:jboss/exported/app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote
ejb:app-send-message-server-ejb/MessageEJB!com.training.MessageEJBRemote

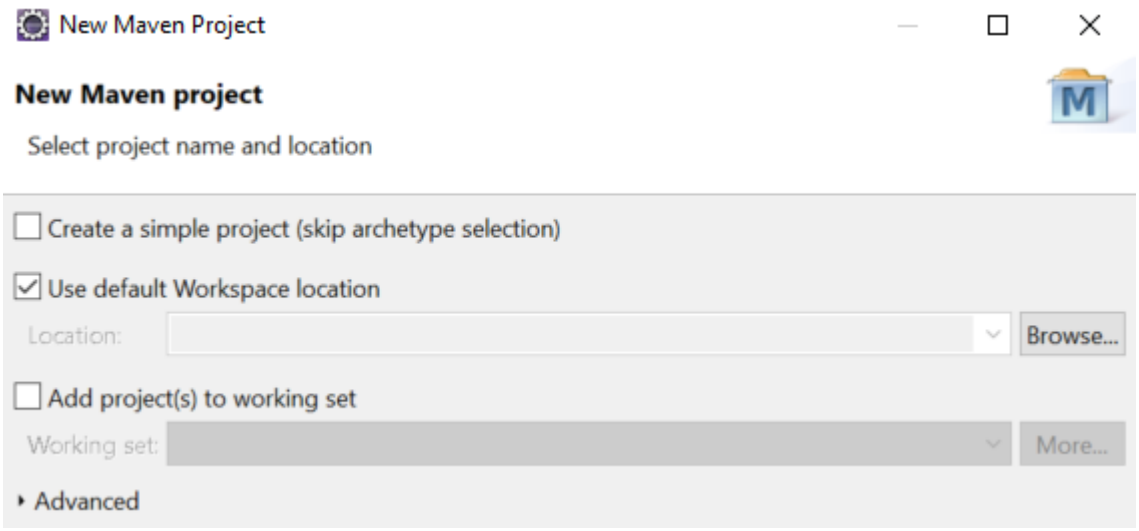
Aller sur http://localhost:9990/console/index.html


localhost:9990/console/index.html
WildFly
Homepage
Deployments
Configuration
Runtime
Patching
Access Control
WildFly Application Server
Deployments
Add and manage deployments
Deploy an Application
Start
Deploy an application to the server
1. Use the 'Add Deployment' wizard to deploy the application
2. Enable the deployment
Configuration
Configure subsystem settings
Create a Datasource
Start
Define a datasource to be used by deployed applications. The proper JDBC driver must be deployed and registered.
1. Select the Datasources subsystem
2. Add a Non-XA or XA datasource
3. Use the 'Create Datasource' wizard to configure the datasource settings
Runtime
Monitor server status
Monitor the Server
Start
View runtime information such as server status, JVM status, and server log files.
1. Select the server
2. View log files or JVM usage
Access Control
Manage user and group permissions for management operations
Assign User Roles
Start
Assign roles to users or groups to determine access to system resources.
1. Add a new user or group
2. Assign one or more roles to that user or group

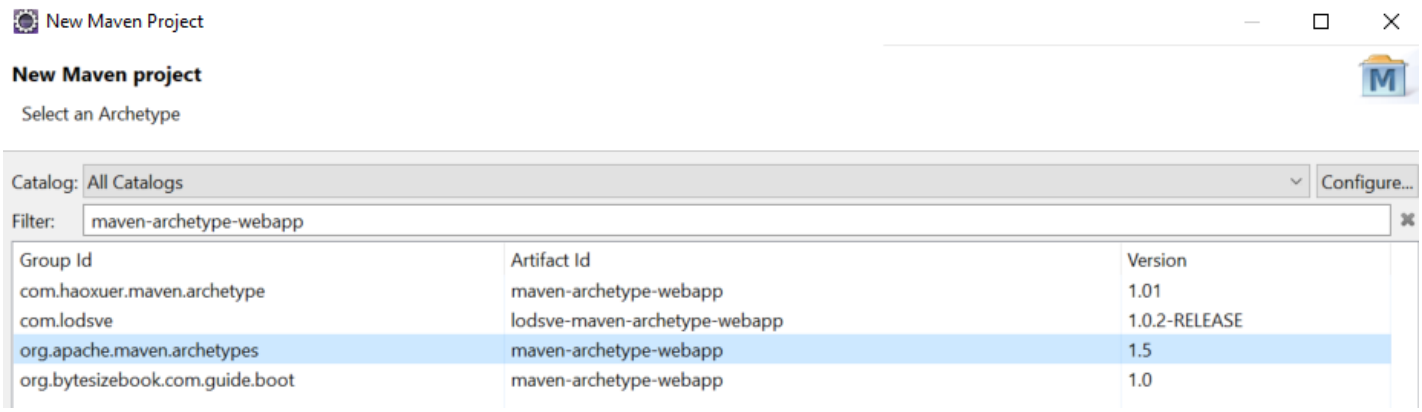

localhost:9990/console/index.html#deployments
WildFly
Homepage
Deployments
Configuration
Runtime
Patching
Access Control
Deployment (1)
Filter by: name or deployment status
app-send-message-server...
Deployments
A deployment represents anything that can be deployed (e.g. an application such as EJB
server.
You can use drag and drop to add new content or replace existing deployments. Simply
same name, the deployment will be replaced, otherwise the deployment will be added.

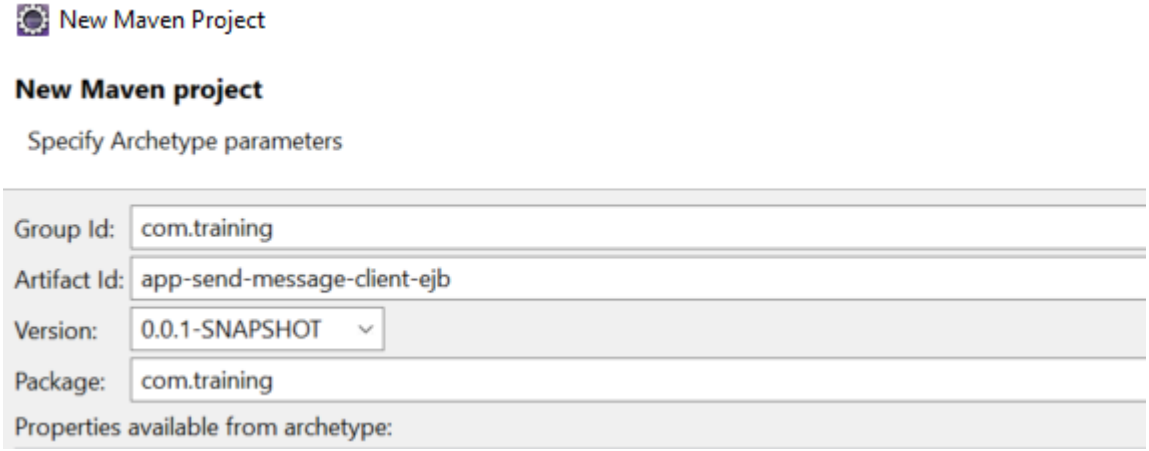
2. *Client Web EJB*

Créer le projet Maven **app-client-web-ejb-sample** de type **war** (archetype=**maven-archetype-webapp**) qui sera notre client EJB.



Taper dans Filter **maven-archetype-webapp**





Créer les dossier **src/main/java**, **src/main/resources**, **src/test/main** et **src/test/resources** s'il n'existe pas.

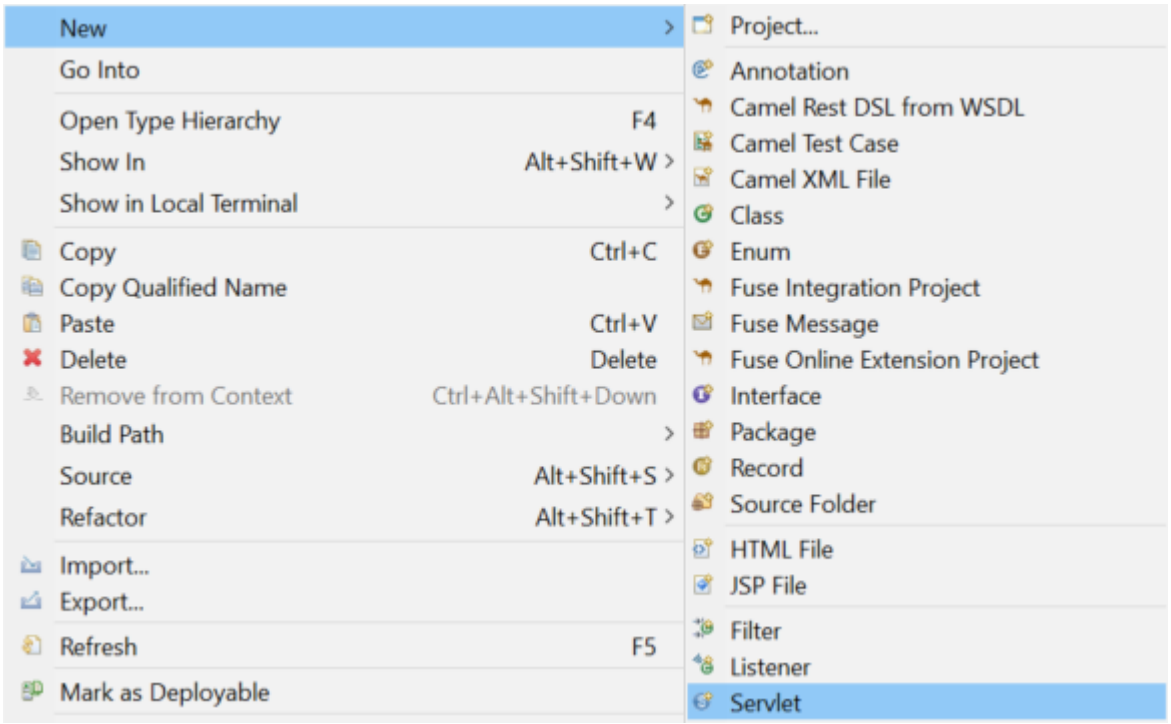
Mettre à jour le fichier **pom.xml** qui devient :

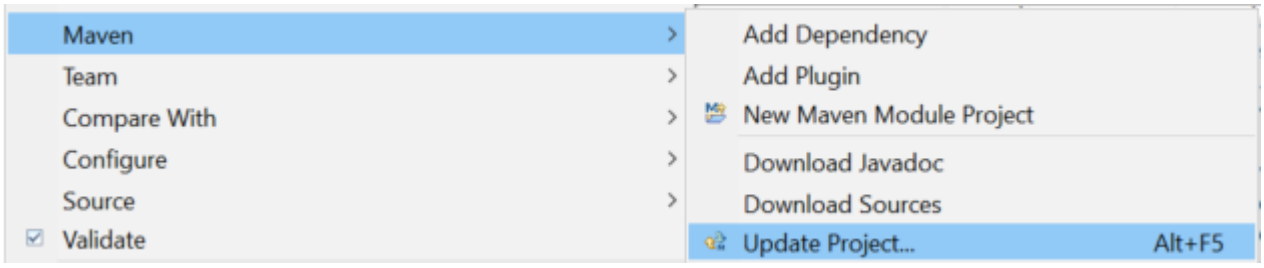
```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.training</groupId>
  <artifactId>app-send-message-client-ejb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>app-send-message-client-ejb Maven Webapp</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <javax.servlet.version>4.0.0</javax.servlet.version>
    <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
    <jstl.version>1.2</jstl.version>
  </properties>

  <dependencies>
    <!-- Debut Java EE API : implémentation -->
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
    <!-- Fin Java EE API : implémentation -->
    <!-- Debut API Servlet : implémentation -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>${javax.servlet.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>${javax.servlet.jsp.version}</version>
      <scope>provided</scope>
    </dependency>
```

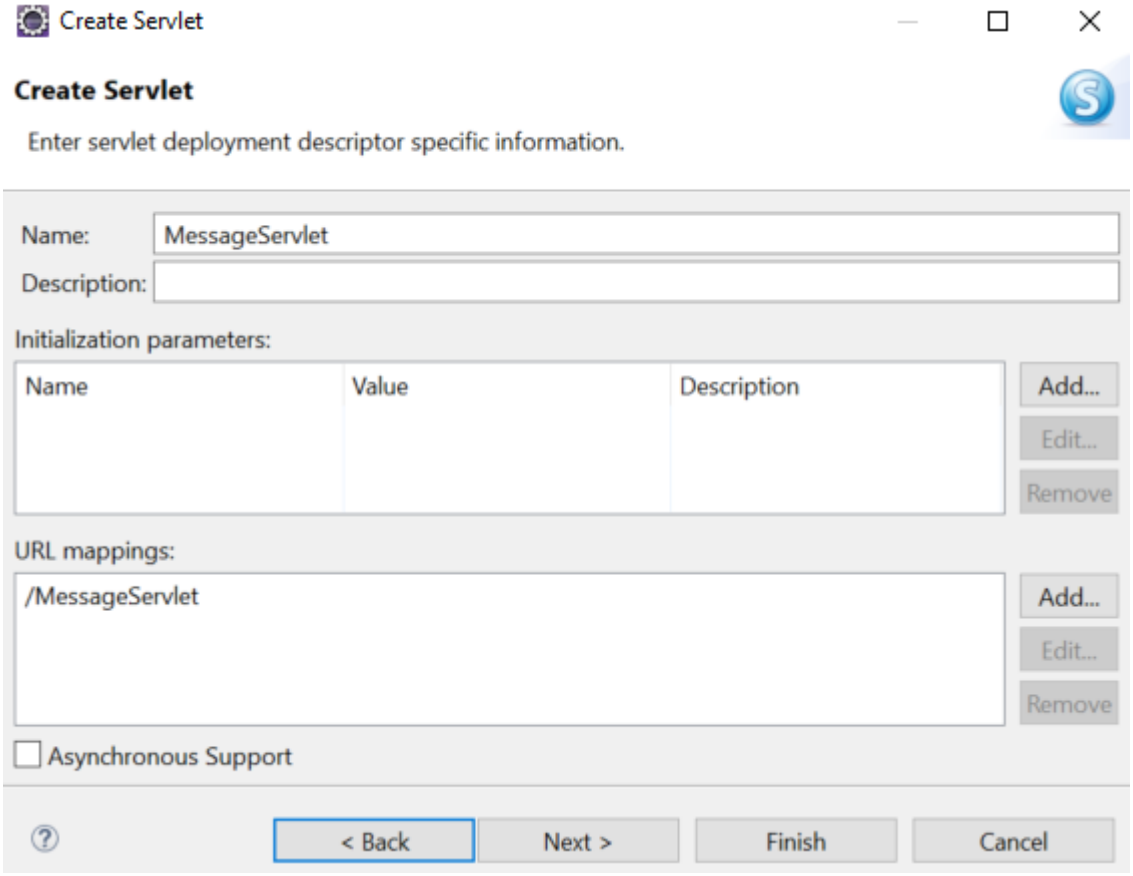
```
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>${jstl.version}</version>
    </dependency>
    <!-- Fin API Servlet : implémentation -->
    <!-- Debut app-send-message-server-ejb -->
    <dependency>
      <groupId>com.training</groupId>
      <artifactId>app-send-message-server-ejb</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <!-- Fin app-send-message-server-ejb -->
  </dependencies>


  <build>
    <finalName>app-send-message-client-ejb</finalName>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven
        defaults (may be moved to parent pom) -->
      <plugins>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.4.0</version>
        </plugin>
        <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.3.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.13.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>3.3.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.4.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>3.1.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
```

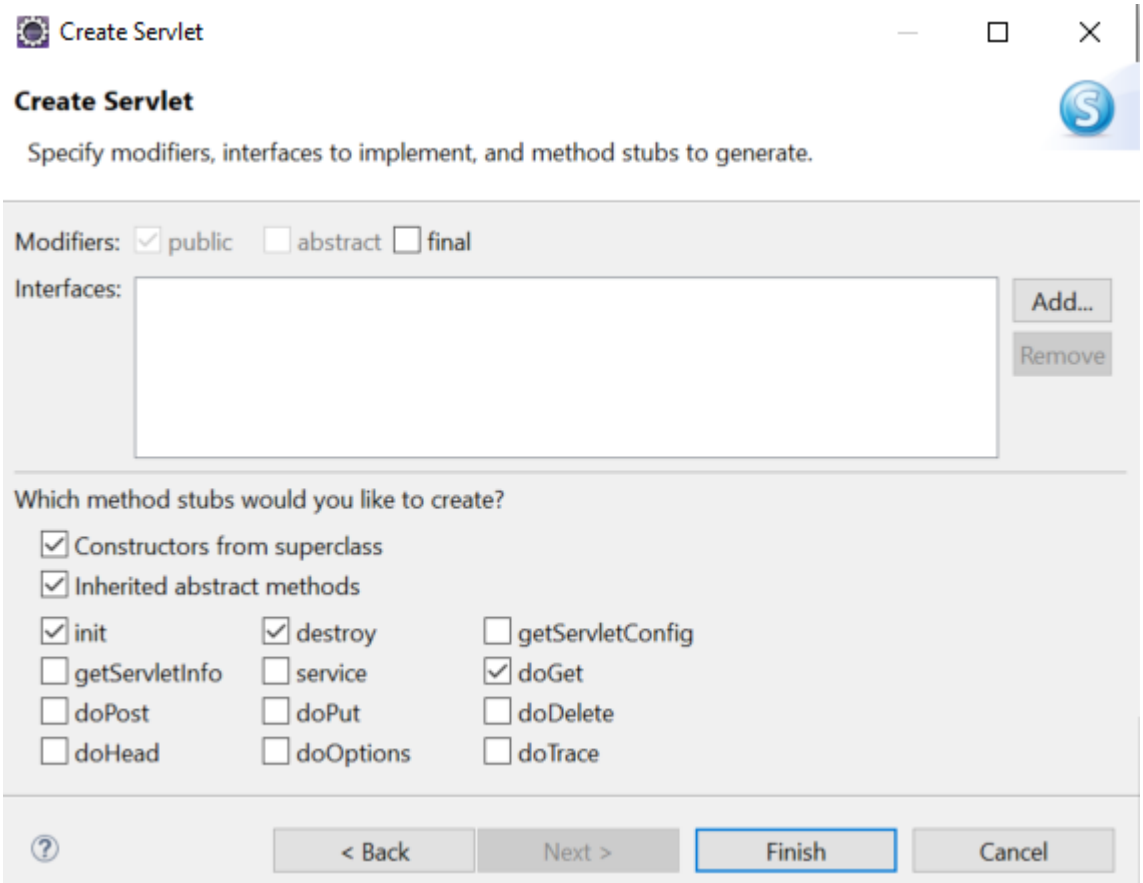
```
            <version>3.1.2</version>
          </plugin>
        </plugins>
      </pluginManagement>
    </build>
</project>
```

Créer la servlet **com.training.MessageServlet** :






Create Servlet
Create Servlet
Enter servlet deployment descriptor specific information.
Name:
MessageServlet
Description:
Initialization parameters:
Name
Value
Description
Add...
Edit...
Remove
URL mappings:
/MessageServlet
Add...
Edit...
Remove
Asynchronous Support
< Back
Next >
Finish
Cancel


Create Servlet
Create Servlet
Specify modifiers, interfaces to implement, and method stubs to generate.
Modifiers:
public
abstract
final
Interfaces:
Add...
Remove
Which method stubs would you like to create?
Constructors from superclass
Inherited abstract methods
init
destroy
getServletConfig
getServletInfo
service
doGet
doPost
doPut
doDelete
doHead
doOptions
doTrace
< Back
Next >
Finish
Cancel

La servlet **com.training.MessageServlet** aura pour contenu initial :

```
package com.training.MessageServlet;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class MessageServlet
 */
@WebServlet("/MessageServlet")
public class MessageServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public MessageServlet() {
      super();
      // TODO Auto-generated constructor stub
   }

   /**
    * @see Servlet#init(ServletConfig)
    */
   public void init(ServletConfig config) throws ServletException {
      // TODO Auto-generated method stub
   }
   /**
    * @see Servlet#destroy()
    */
   public void destroy() {
      // TODO Auto-generated method stub
   }
   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
      // TODO Auto-generated method stub
      response.getWriter().append("Served at: ").append(request.getContextPath());
   }

}
```

En utilisant le Context JNDI la classe **com.training.MessageServlet** peut devenir :

```
package com.training;

import java.io.IOException;

import javax.ejb.EJB;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class MessageServlet
 */
@WebServlet("/MessageServlet")
public class MessageServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public MessageServlet() {
      super();
      // TODO Auto-generated constructor stub
   }

   /**
    * @see Servlet#init(ServletConfig)
    */
   public void init(ServletConfig config) throws ServletException {
      // TODO Auto-generated method stub
   }

   /**
    * @see Servlet#destroy()
    */
   public void destroy() {
      // TODO Auto-generated method stub
   }

   private String getMessageContext() {
```

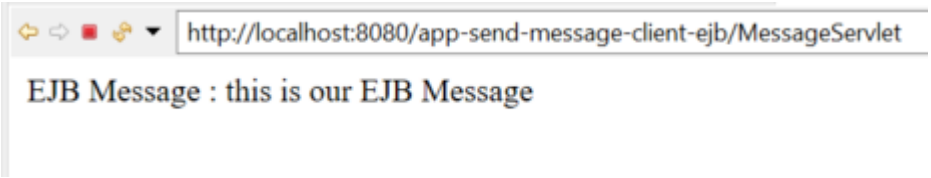
```
        String message = null;
        Context initialContext = null;
        try {
            initialContext = new InitialContext();
            // instanciation MessageEJB
            MessageEJB messageEJB = (MessageEJB) initialContext
                    .lookup("java:module/MessageEJB!com.training.MessageEJB");
            message = messageEJB.sendMessage();
            System.out.println("getMessageContext --> message : " + message);
        } catch (Exception e) {
            System.out.println("Erreur lors de l'execution, Exception : " + e);
        } finally {
            try {
                if (initialContext != null) {
                    initialContext.close();
                }
            } catch (NamingException ex) {
                System.out.println("Erreur lors de l'execution, Exception : " + ex);
            }
        }
        return message;
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     *      response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String message = getMessageContext();
        response.getWriter().print("EJB Message : " + message);
    }
}
```

En utilisant l'annotation **@EJB** la classe **com.training.MessageServlet** peut devenir :

```
package com.training;

import java.io.IOException;
import javax.ejb.EJB;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class MessageServlet
 */
@WebServlet("/MessageServlet")
public class MessageServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @EJB
   private MessageEJB messageEJB;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public MessageServlet() {
      super();
      // TODO Auto-generated constructor stub
   }

   /**
    * @see Servlet#init(ServletConfig)
    */
   public void init(ServletConfig config) throws ServletException {
      // TODO Auto-generated method stub
   }

   /**
    * @see Servlet#destroy()
    */
   public void destroy() {
      // TODO Auto-generated method stub
   }
```

```
  private String getMessage() {
    String message = messageEJB.sendMessage();
    System.out.println("getMessage --> message : " + message);
    return message;
  }

  private String getMessageContext() {
    String message = null;
    Context initialContext = null;
    try {
      initialContext = new InitialContext();
      // instanciation MessageEJB
      MessageEJB messageEJB = (MessageEJB) initialContext
          .lookup("java:module/MessageEJB!com.training.MessageEJB");
      message = messageEJB.sendMessage();
      System.out.println("getMessageContext --> message : " + message);
    } catch (Exception e) {
      System.out.println("Erreur lors de l'execution, Exception : " + e);
    } finally {
      try {
        if (initialContext != null) {
          initialContext.close();
        }
      } catch (NamingException ex) {
        System.out.println("Erreur lors de l'execution, Exception : " + ex);
      }
    }
    return message;
  }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
   *      response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    String message = getMessage();
    response.getWriter().print("EJB Message : " + message);
  }
}
```

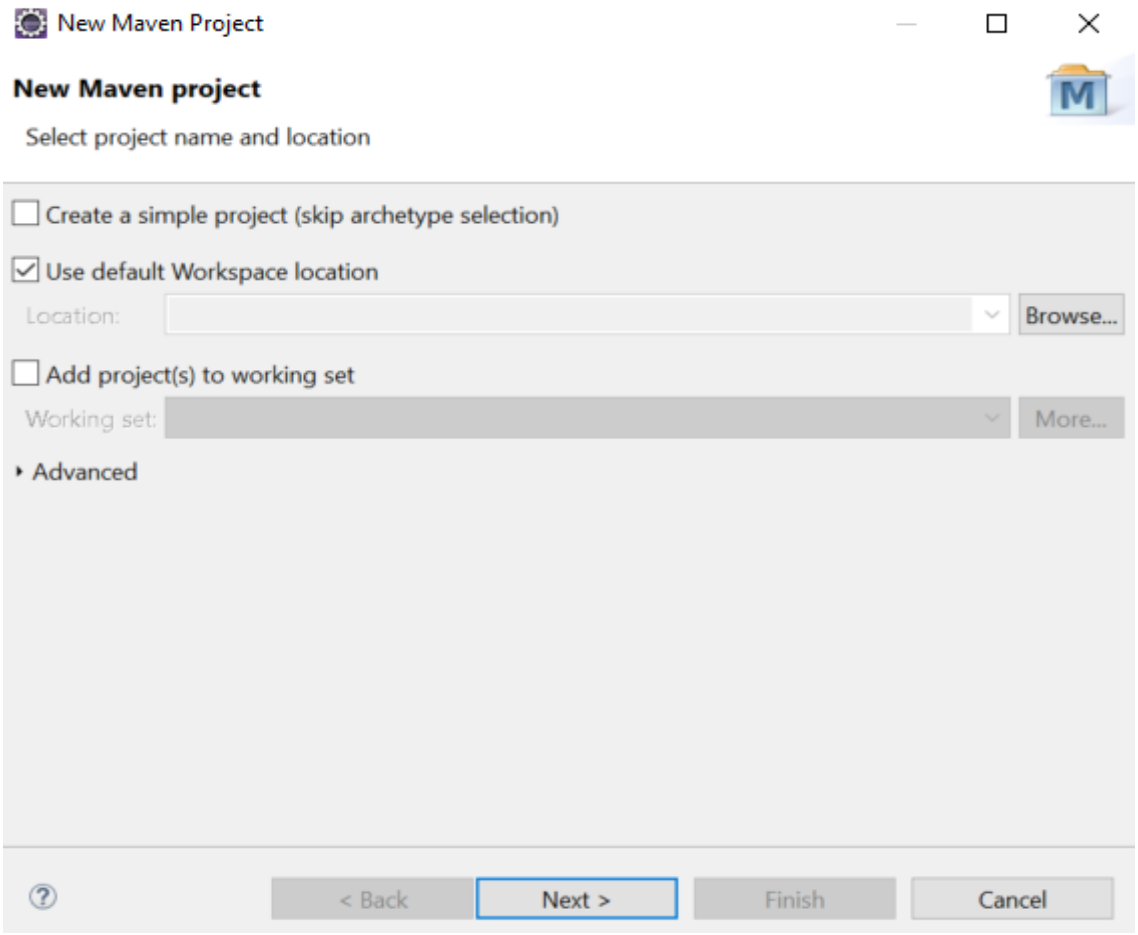
Le lancement de l'application nous donne sur http://localhost:8080/app-send-message-client-ejb/MessageServlet



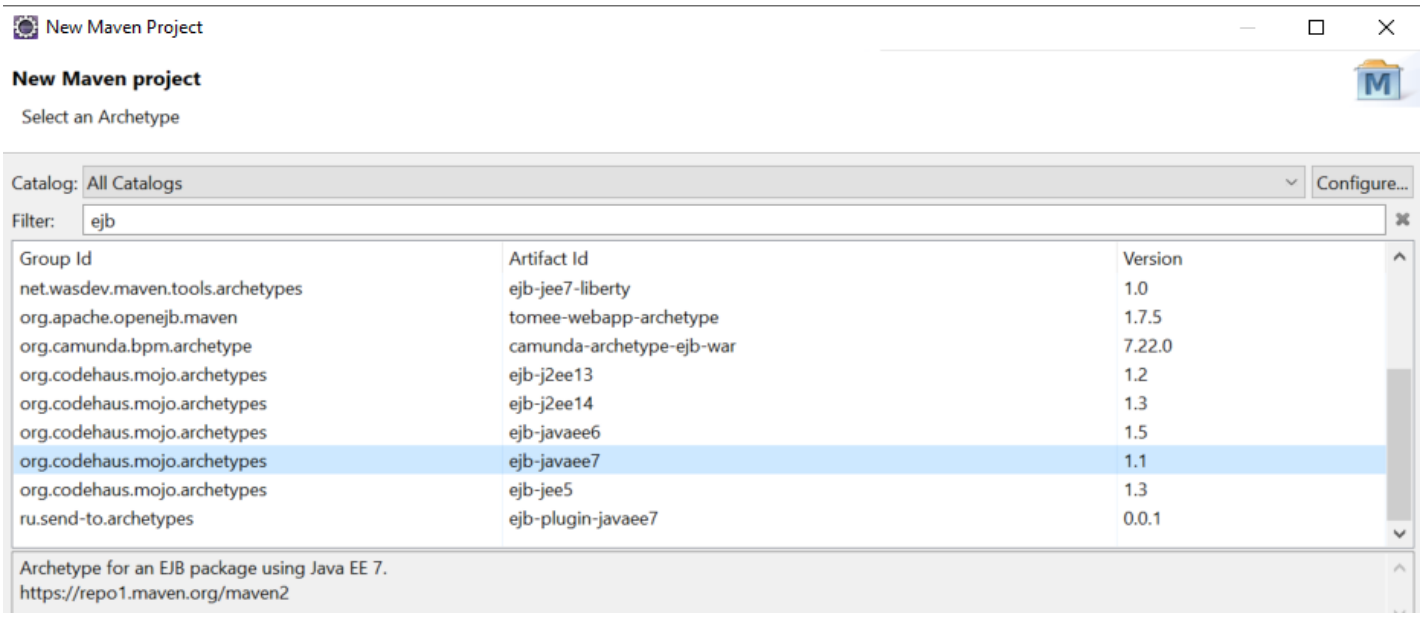
V) Les différents états d'un EJB

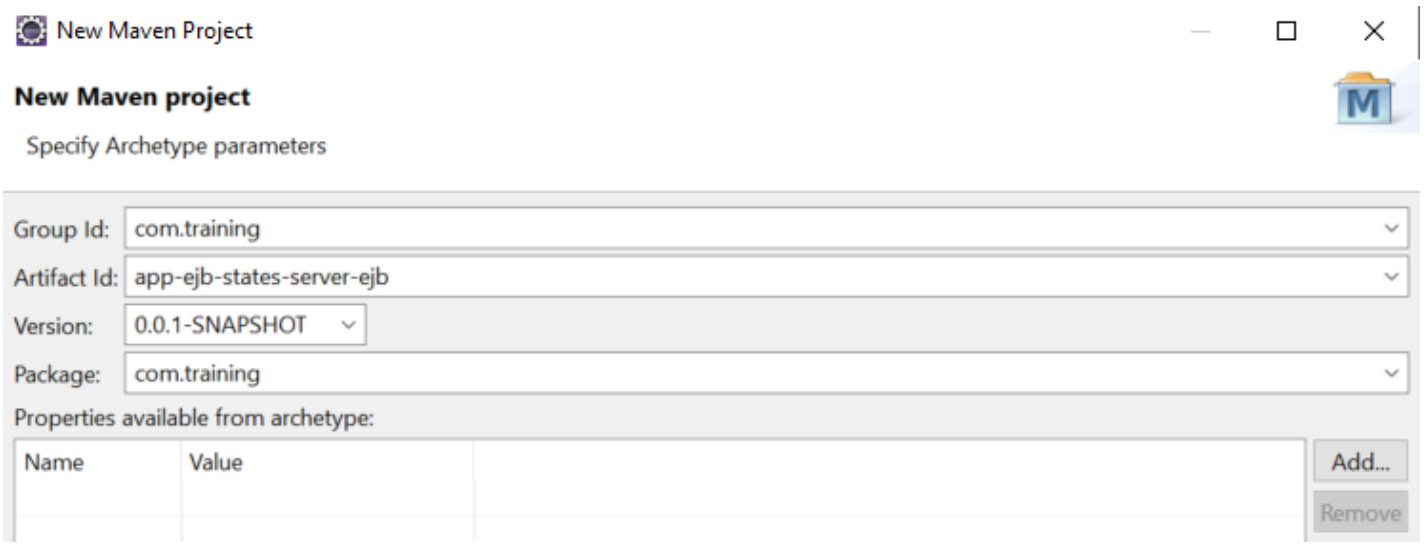
1. *Création d'un Serveur EJB*

Créer le projet **app-ejb-states-server-ejb** qui sera notre Serveur EJB.



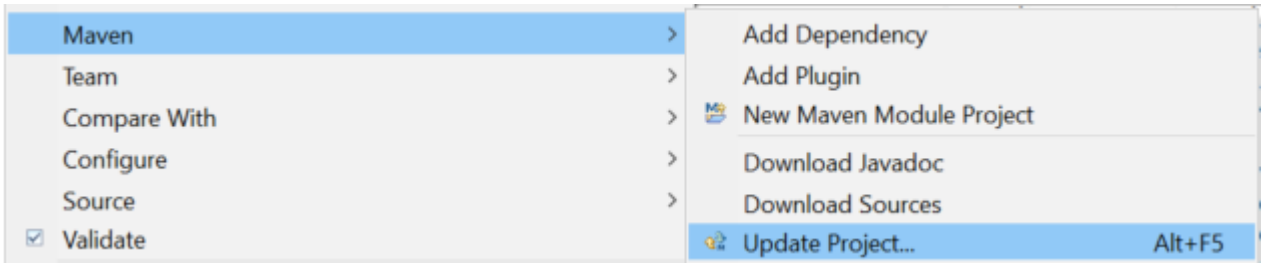
Taper sur Filter **ejb**.

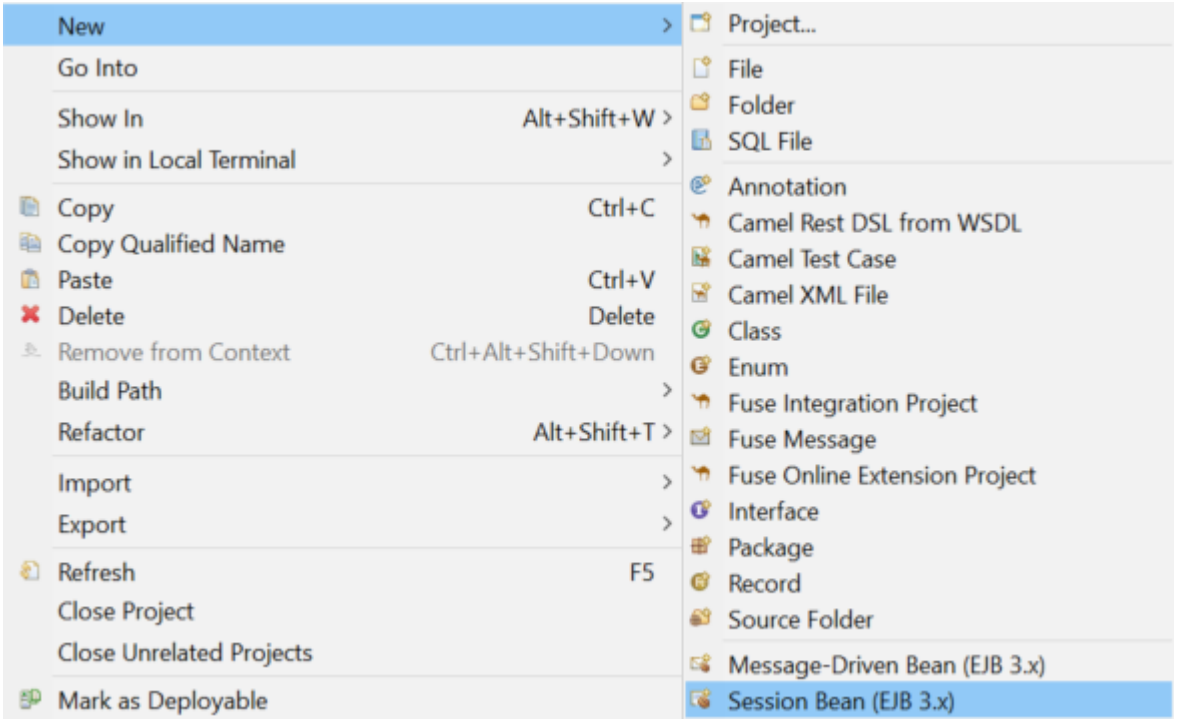




Ajouter la balise **<pluginManagement>** avant la balise **<plugins>** dans le **pom.xml**.

Créer les dossier **src/test/main** et **src/test/resources** s'il n'existe pas.



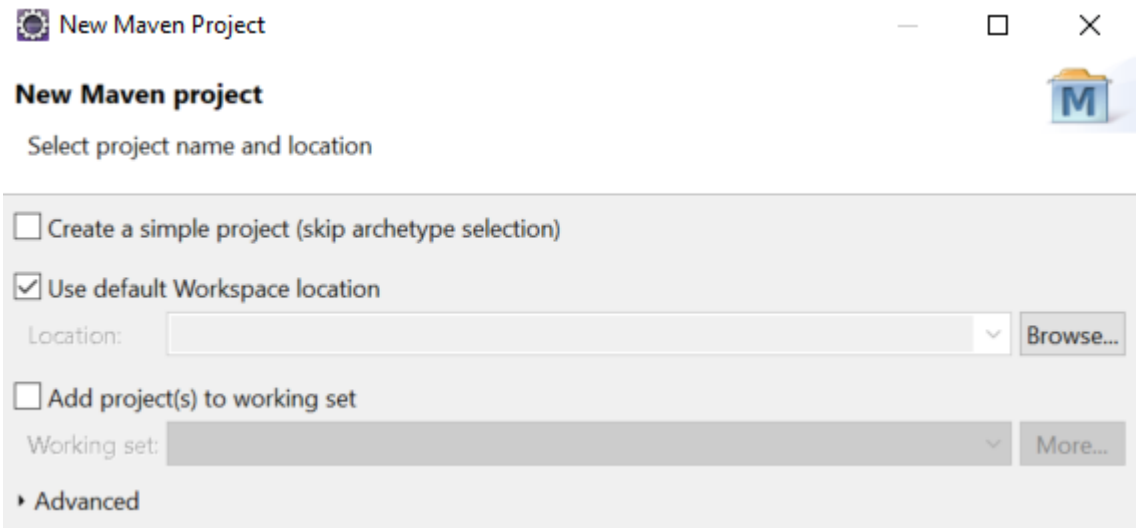


La classe **com.training.ejb.CounterBean** aura comme contenu :

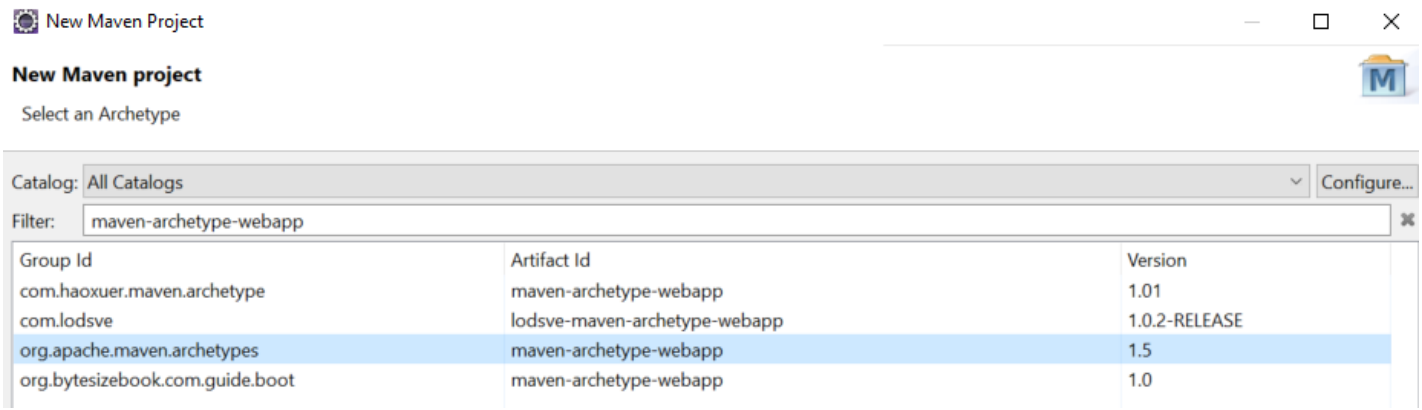
```
package com.training.ejb;

import javax.ejb.Stateless;
import java.util.concurrent.atomic.AtomicInteger;
import javax.ejb.LocalBean;

/**
 *
 * @author El Hadji Gaye
 */
@Stateless
@LocalBean
public class CounterBean {

    private AtomicInteger number = new AtomicInteger(0);

    public int getNumber() {
        return number.get();
    }

    public void increment() {
        this.number.getAndIncrement();
    }
}
```

2. *Création d'un Client Web EJB*

Créer le projet Maven **app-ejb-states-server-client** de type **war** (archetype=**maven-archetype-webapp**) qui sera notre client EJB.



Taper dans Filter **maven-archetype-webapp**



Créer les dossier **src/main/java**, **src/main/resources**, **src/test/main** et **src/test/resources** s'ils n'existent pas.

Mettre à jour le fichier **pom.xml** qui devient :

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.training</groupId>
  <artifactId>app-ejb-states-server-client</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>app-ejb-states-server-client Maven Webapp</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <javax.servlet.version>4.0.0</javax.servlet.version>
    <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
    <jstl.version>1.2</jstl.version>
  </properties>

  <dependencies>
    <!-- Debut Java EE API : implémentation -->
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
    <!-- Fin Java EE API : implémentation -->
    <!-- Debut API Servlet : implémentation -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>${javax.servlet.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>${javax.servlet.jsp.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>jstl</groupId>
```

```
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
      </dependency>
      <!-- Fin API Servlet : implémentation -->
      <!-- Debut app-ejb-states-server-ejb -->
      <dependency>
        <groupId>com.training</groupId>
        <artifactId>app-ejb-states-server-ejb</artifactId>
        <version>0.0.1-SNAPSHOT</version>
      </dependency>
      <!-- Fin app-ejb-states-server-ejb -->
    </dependencies>


    <build>
      <finalName>app-ejb-states-server-client</finalName>
      <pluginManagement><!-- lock down plugins versions to avoid using Maven
          defaults (may be moved to parent pom) -->
        <plugins>
          <plugin>
            <artifactId>maven-clean-plugin</artifactId>
            <version>3.4.0</version>
          </plugin>
          <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
          <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>3.3.1</version>
          </plugin>
          <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.13.0</version>
          </plugin>
          <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.3.0</version>
          </plugin>
          <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.4.0</version>
          </plugin>
          <plugin>
            <artifactId>maven-install-plugin</artifactId>
            <version>3.1.2</version>
          </plugin>
          <plugin>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>3.1.2</version>
          </plugin>
```

```
        </plugins>
      </pluginManagement>
    </build>
</project>
```

Créer la servlet **com.training.PageOne** avec le contenu :

```
package com.training;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.training.ejb.CounterBean;

/**
 * Servlet implementation class PageOne
 */
@WebServlet("/PageOne")
public class PageOne extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @EJB
   private CounterBean counterBean;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public PageOne() {
      super();
   }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    *      response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      counterBean.increment();
      out.println("PageOne --> number : " + counterBean.getNumber());
   }
}
```

Créer la servlet **com.training.PageTwo** avec le contenu :

```
package com.training;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.training.ejb.CounterBean;

/**
 * Servlet implementation class PageOne
 */
@WebServlet("/PageTwo")
public class PageTwo extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @EJB
   private CounterBean counterBean;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public PageTwo() {
      super();
   }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    *      response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      counterBean.increment();
      out.println("PageTwo --> number : " + counterBean.getNumber());
   }
}
```

Créer la servlet **com.training.PageThree** avec le contenu :

```
package com.training;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.training.ejb.CounterBean;

/**
 * Servlet implementation class PageOne
 */
@WebServlet("/PageThree")
public class PageThree extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @EJB
   private CounterBean counterBean;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public PageThree() {
      super();
   }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    *      response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      counterBean.increment();
      out.println("PageThree --> number : " + counterBean.getNumber());
   }
}
```

3. *L'état @Stateless*

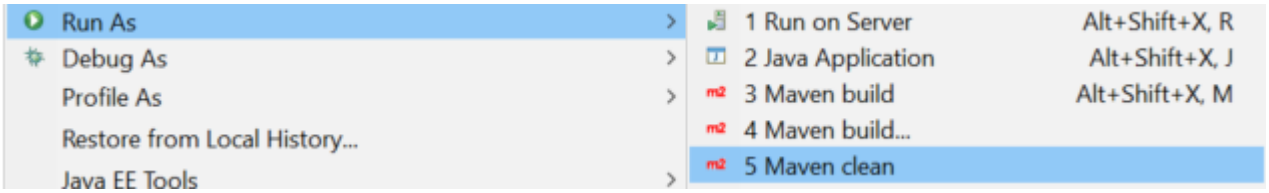
Un bean de session stateless (sans état) ne conserve pas d'état conversationnel avec le client. Lorsqu'un client appelle les méthodes d'un bean sans état, les variables d'instance du bean peuvent contenir un état spécifique à ce client, mais uniquement pendant la durée de l'appel. Lorsque la méthode est terminée, l'état spécifique au client ne doit pas être conservé. Les clients peuvent cependant modifier l'état des variables d'instance dans les beans sans état regroupés, et cet état est conservé jusqu'à la prochaine invocation du bean sans état regroupé. Sauf pendant l'invocation de la méthode, toutes les instances d'un bean sans état sont équivalentes, ce qui permet au conteneur EJB d'attribuer une instance à n'importe quel client. Autrement dit, l'état d'un bean de session sans état doit s'appliquer à tous les clients.

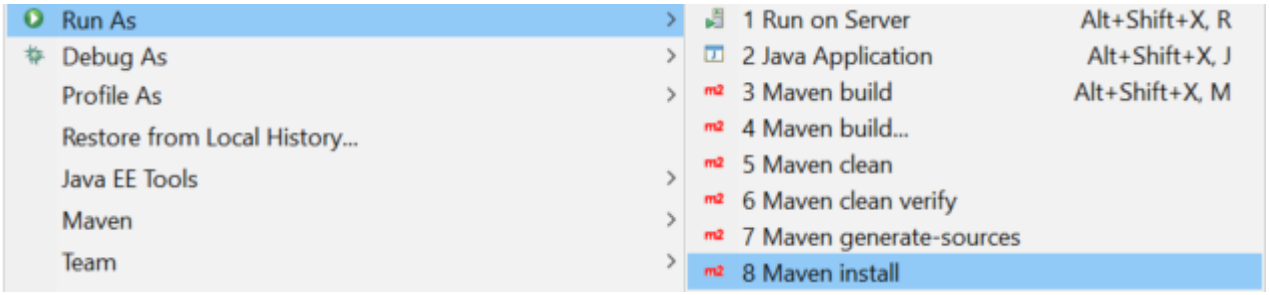
Étant donné qu'ils peuvent prendre en charge plusieurs clients, les beans de session sans état peuvent offrir une meilleure évolutivité pour les applications qui nécessitent un grand nombre de clients. En règle générale, une application nécessite moins de beans de session sans état que de beans de session avec état pour prendre en charge le même nombre de clients.

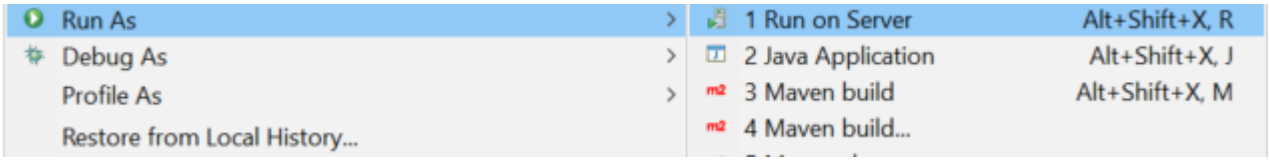
Un bean de session (stateless) sans état peut implémenter un service Web, mais un bean de session avec état ne le peut pas.

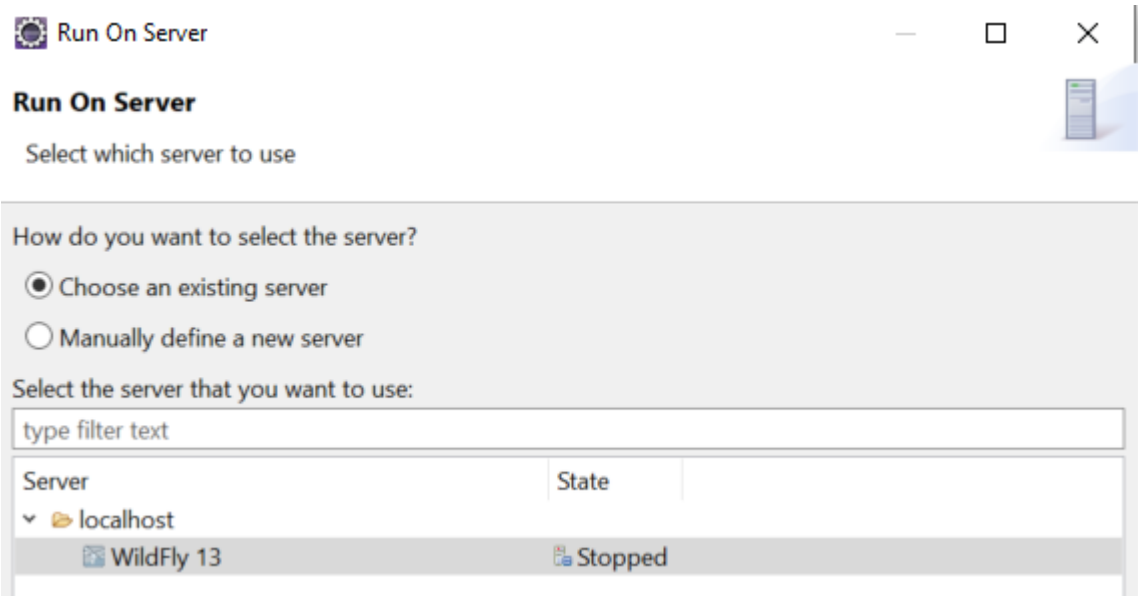
La classe La classe **app-ejb-states-server-ejb/src/main/java/com/training/ejb/CounterBean.java** aura comme contenu :

```
package com.training.ejb;

import javax.ejb.Stateless;
import java.util.concurrent.atomic.AtomicInteger;
import javax.ejb.LocalBean;

/**
 *
 * @author El Hadji Gaye
 */
@Stateless
@LocalBean
public class CounterBean {

    private AtomicInteger number = new AtomicInteger(0);

    public int getNumber() {
        return number.get();
    }

    public void increment() {
        this.number.getAndIncrement();
    }
}
```

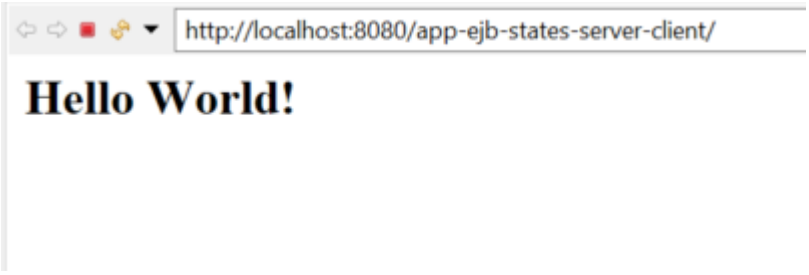

Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean


Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven clean verify
7 Maven generate-sources
8 Maven install

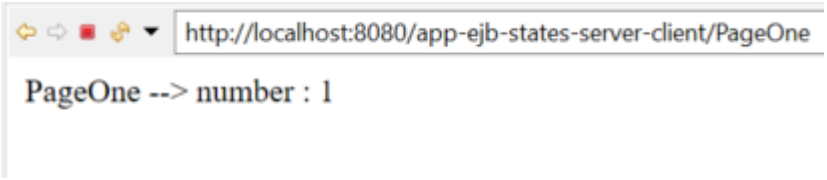

Run As
Debug As
Profile As
Restore from Local History...
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
WildFly 13
Stopped

Le lancement de l'application nous donne sur http://localhost:8080/app-ejb-states-server-client/



http://localhost:8080/app-ejb-states-server-client/PageOne



http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageTwo

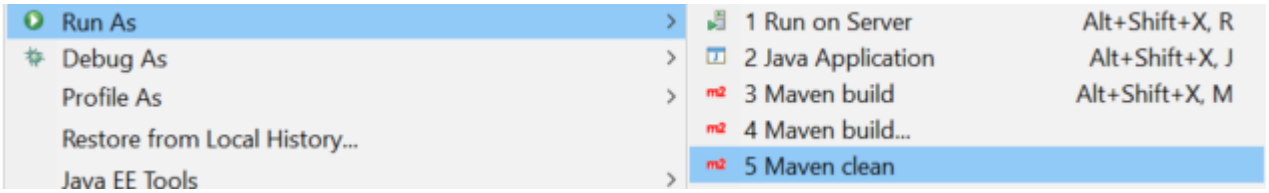
http://localhost:8080/app-ejb-states-server-client/PageThree

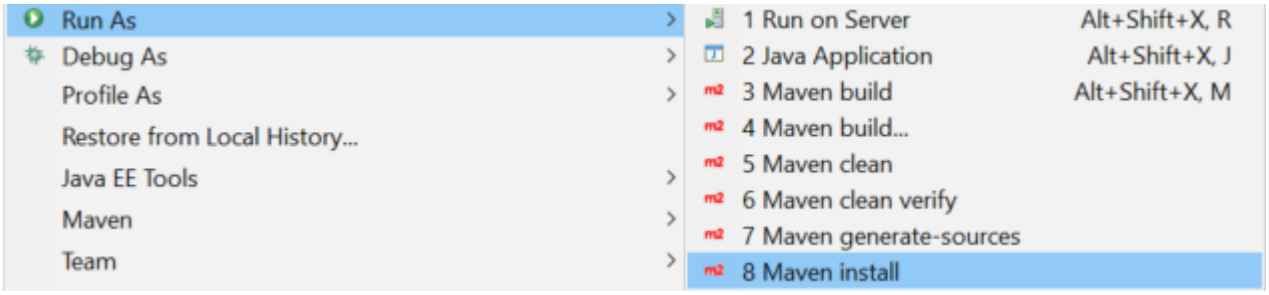
4. *L'état @Stateful*

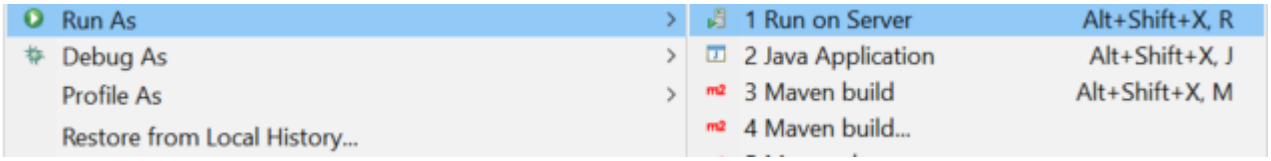
L'état d'un objet se compose des valeurs de ses variables d'instance. Dans un bean de session stateful (avec état), les variables d'instance représentent l'état d'une session client/bean unique. Car le client interagit avec son bean. Comme son nom l'indique, un bean de session est similaire à une session interactive. Un bean de session n'est pas partagé ; il ne peut avoir qu'un seul client, de la même manière qu'une session interactive ne peut avoir qu'un seul utilisateur. Lorsque le client se termine, son bean de session semble se terminer et n'est plus associé au client. L'état est conservé pendant toute la durée de la session client/bean. Si le client supprime le bean, la session se termine et l'état disparaît. Cette nature transitoire de l'état n'est cependant pas un problème, car lorsque la conversation entre le client et le bean se termine, il n'est pas nécessaire de conserver l'état.

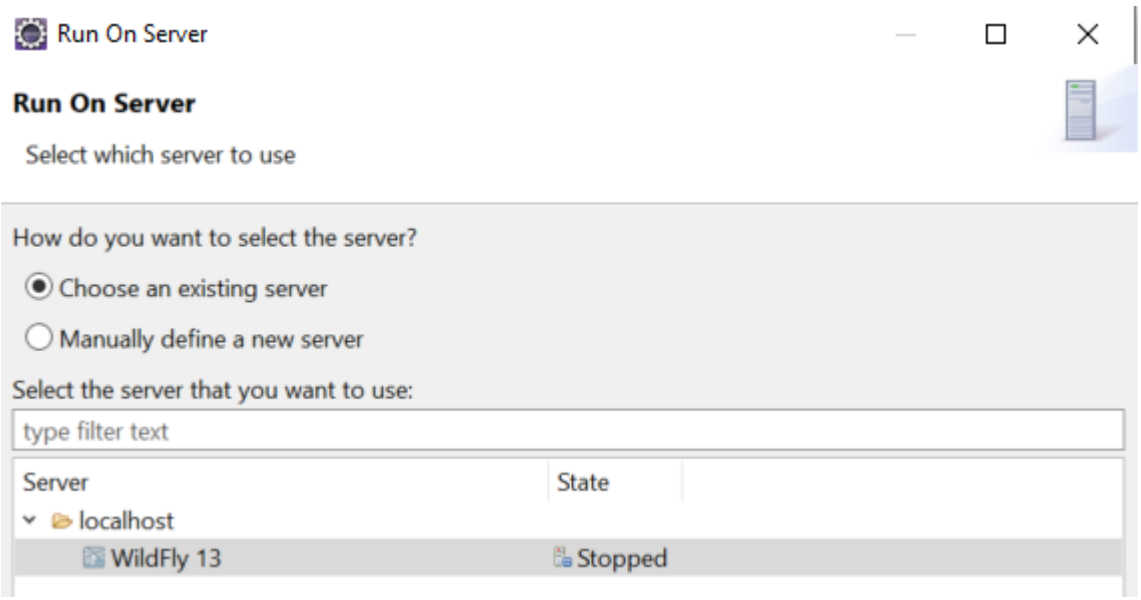
La classe **app-ejb-states-server-ejb/src/main/java/com/training/ejb/CounterBean.java** peut avoir comme contenu :

```
package com.training.ejb;

import java.util.concurrent.atomic.AtomicInteger;

import javax.ejb.LocalBean;
import javax.ejb.Stateful;

/**
 *
 * @author El Hadji Gaye
 */
@Stateful
@LocalBean
public class CounterBean {

    private AtomicInteger number = new AtomicInteger(0);

    public int getNumber() {
        return number.get();
    }

    public void increment() {
        this.number.getAndIncrement();
    }
}
```

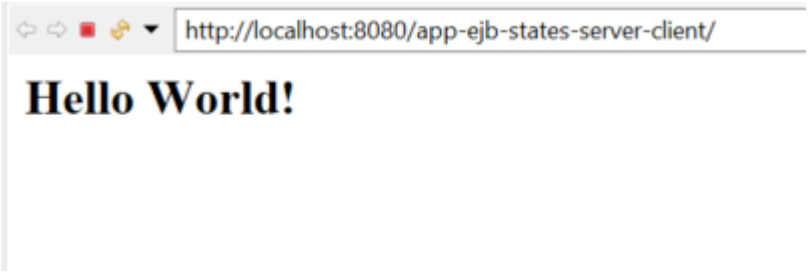

Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean


Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven clean verify
7 Maven generate-sources
8 Maven install

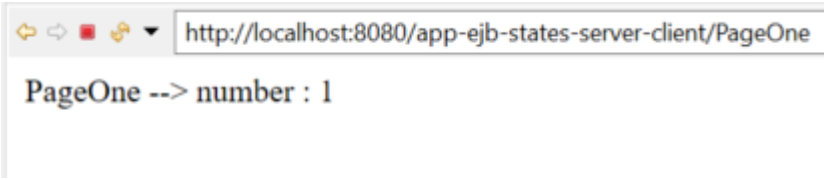

Run As
Debug As
Profile As
Restore from Local History...
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
WildFly 13
Stopped

Le lancement de l'application nous donne sur http://localhost:8080/app-ejb-states-server-client/



http://localhost:8080/app-ejb-states-server-client/PageOne



http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageTwo

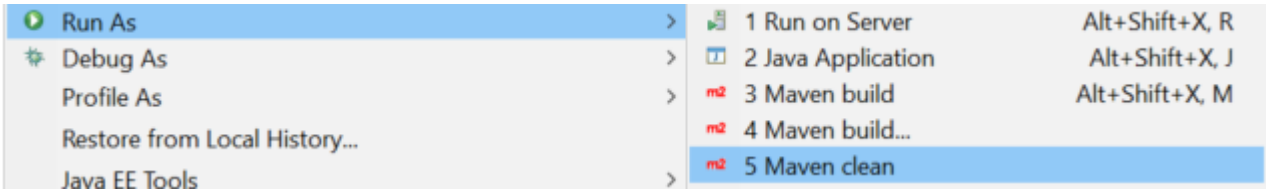
http://localhost:8080/app-ejb-states-server-client/PageThree

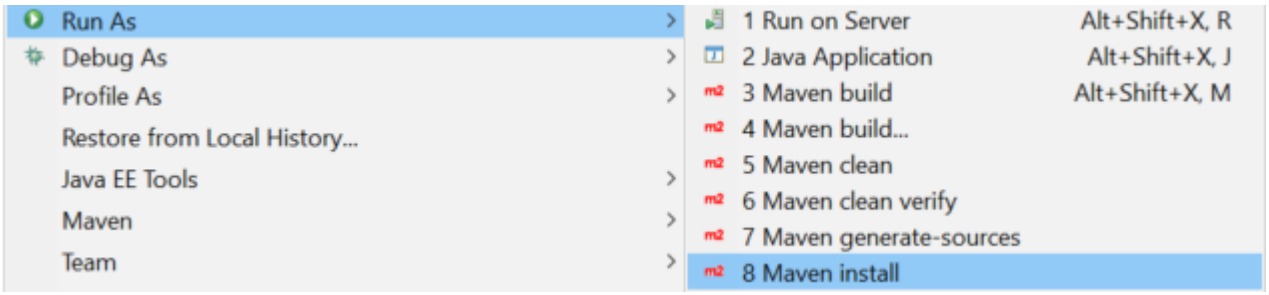
5. *L'état @Singleton*

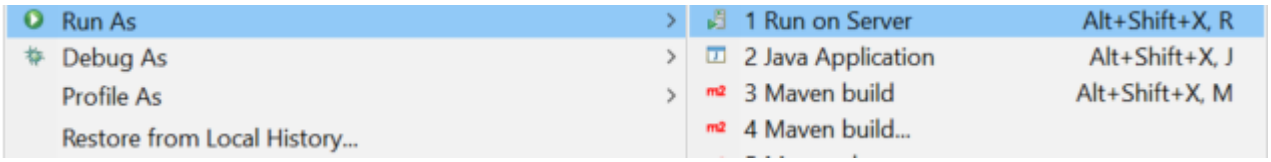
Un bean de session singleton est instancié une fois par application et existe pendant tout le cycle de vie de l'application. Les beans de session singleton sont conçus pour les situations dans lesquelles une instance de bean d'entreprise unique est partagée et accessible simultanément par les clients. Les beans de session singleton offrent des fonctionnalités similaires aux beans de session sans état, mais diffèrent d'eux dans le sens où il n'y a qu'un seul bean de session singleton par application, par opposition à un pool de beans de session sans état, dont chacun peut répondre à une demande client. Comme les beans de session sans état, les beans de session singleton peuvent implémenter des points de terminaison de service Web. Les beans de session singleton conservent leur état entre les invocations client, mais ne sont pas tenus de conserver leur état en cas de panne ou d'arrêt du serveur. Les applications qui utilisent un bean de session singleton peuvent spécifier que le singleton doit être instancié au démarrage de l'application, ce qui permet au singleton d'effectuer des tâches d'initialisation pour l'application. Le singleton peut également effectuer des tâches de nettoyage à l'arrêt de l'application, car il fonctionnera tout au long du cycle de vie de l'application.

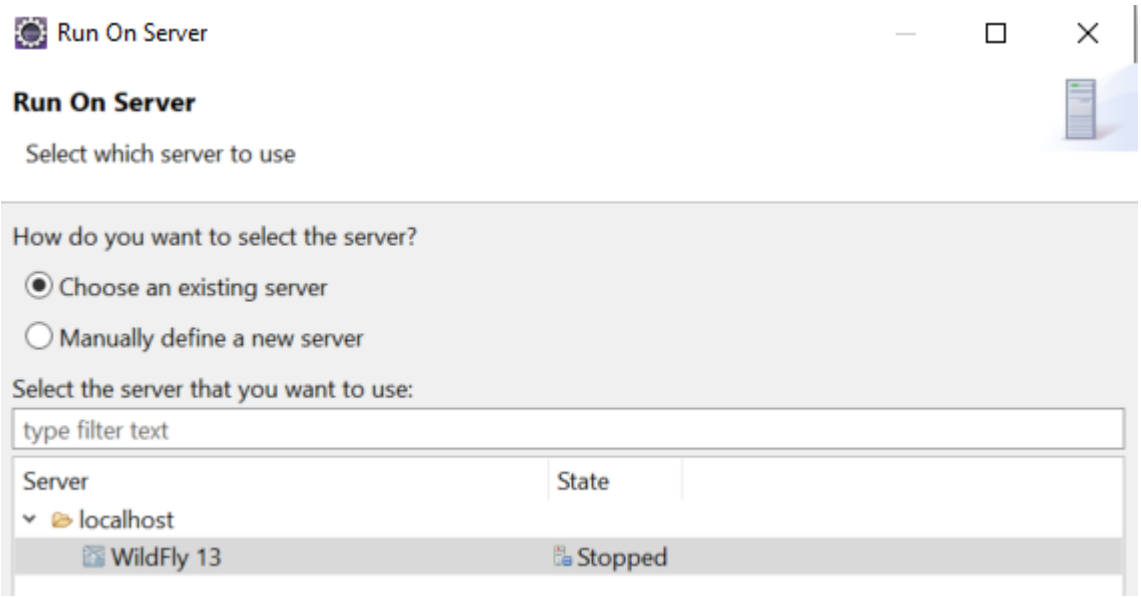
La classe **app-ejb-states-server-ejb/src/main/java/com/training/ejb/CounterBean.java** peut avoir comme contenu :

```
package com.training.ejb;

import java.util.concurrent.atomic.AtomicInteger;

import javax.ejb.LocalBean;
import javax.ejb.Singleton;

/**
 *
 * @author El Hadji Gaye
 */
@Singleton
@LocalBean
public class CounterBean {

   private AtomicInteger number = new AtomicInteger(0);

   public int getNumber() {
      return number.get();
   }

   public void increment() {
      this.number.getAndIncrement();
   }
}
```

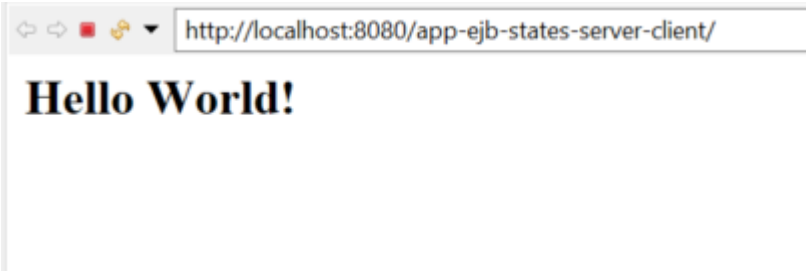

Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean


Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven clean verify
7 Maven generate-sources
8 Maven install

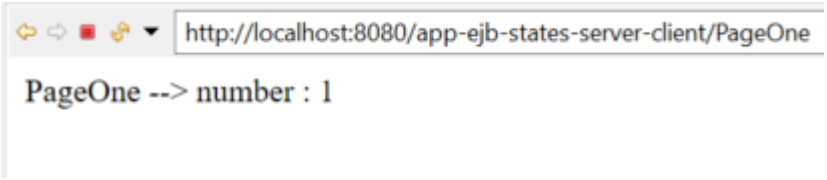

Run As
Debug As
Profile As
Restore from Local History...
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
WildFly 13
Stopped

Le lancement de l'application nous donne sur http://localhost:8080/app-ejb-states-server-client/



http://localhost:8080/app-ejb-states-server-client/PageOne



http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageOne

http://localhost:8080/app-ejb-states-server-client/PageTwo

http://localhost:8080/app-ejb-states-server-client/PageThree