www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Spring Framework 2.5

Training Kit

Naoufel EL HAJ – Walid MELLOULI

Table des matières

A propos des auteurs 6

Atelier 1 - Spring IDE 8

Installation 8
Projet de nature Spring 10
Fichier de configuration Spring 13
Application Spring HelloWorld 16
Ajout des librairies Spring 17
Exécution du code 21
Refactoring 22

Atelier 2 - Inversion of Contrôle (IoC) 25

Principe 25
Exemple d’utilisation 27
Amélioration 31

Atelier 3 - Programmation Orientée Aspect (AOP) 36

Principe 36
Lexique 37
Exemple d’utilisation 38

Atelier 4 - JPA managé par Spring 44

Principe 44
Exemple d’utilisation 44
Création du projet java 44
Génération des Entités JPA 51
Couche DAO 53
Couche Métier 55
Configuration du projet 56
Exécution du projet 60

Atelier 5 - Spring MVC 62
Principe 62
Exemple d'utilisation 63
Ajout d'un contrôleur 66
Configuration du « DispatcherServlet » 67
Fichier de configuration 68
Ajout de la requête (page redirect.jsp) 72
Ajout de la vue 72
Exécution de l'application 73

Atelier 6 - Intégration Spring dans JSF 76
Principe 76
Exemple d'utilisation 76
Création d'un projet web JSF de nature Spring : 76
Configuration de web.xml 79
Configuration de faces-config.xml 80
Création de la classe Etudiant 80
Création de applicationContext.xml 81
Création d'une page web 81
Exécution 82

Atelier 7 - Spring Security 84
Principe 84
Exemple d'utilisation 86
Configuration du web.xml 87
Configuration de app-security.xml 88
Une configuration de base 89
Configuration avec JDBC provider 90
Authentification par formulaire 91
Test de l'application 92

Atelier 8 - Spring .NET 95

Principe 95

Exemple d'utilisation 95

Création du projet 95

Configuration de Spring.Net 96

Ajout des classes 98

Définition des objets 99

Récupération de l'objet 100

Exécution 100

A propos des auteurs

Naoufel EL HAJ élève ingénieur en **Génie Logiciel** à l'Institut National des Sciences Appliquées et de Technologie de Tunis.
Passionné par le développement web, il a contribué au développement des plusieurs sites web en Java J2EE, PHP et flash pour des entreprises et des associations Tunisiennes (en freelance).
Certifié **SCJP**, **ASP.NET**, **WCF** et **CCNA**.
Membre du club **LibertySoft** et **MIC** à l'INSAT.

Walid MELLOULI élève ingénieur en **Génie Logiciel** à l'Institut National des Sciences Appliquées et de Technologie de Tunis.
Certifié **SCJP**, **SCWCD**, **ASP.Net** et **WCF** et **CCNA**.
Contributeur **Sourgeforge.net**, développeur du firewall open source **BadTuxWall Linux Firewall**.
Membre du club **LibertySoft** et **MIC** à l'INSAT.
Citation « *Be the master of your computer, not the slave* »

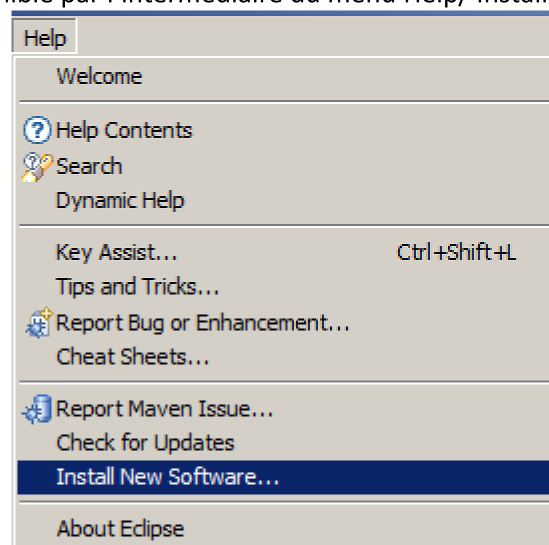
Atelier 1: Spring IDE

Atelier 1 - Spring IDE

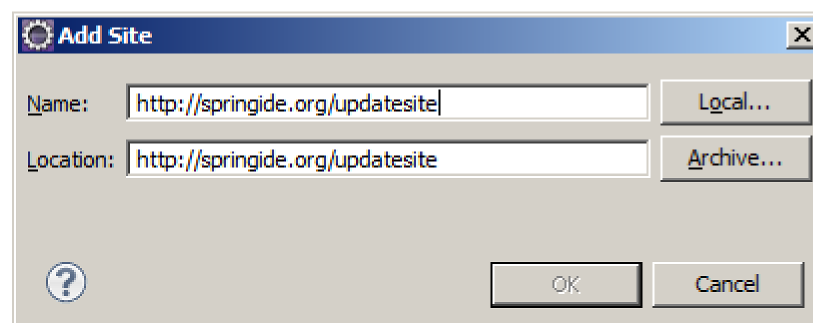
Spring IDE est un plug-in qui permet de faciliter le travail avec Spring Framework dans Eclipse. Il offre des fonctionnalités comme par exemple des wizards de génération des fichiers de configuration de Spring, l'auto-complétion dans ces fichiers (XML), visualisation graphique des beans Spring et de leurs dépendances, etc.

Installation

Afin d'installer Spring IDE dans Eclipse, il est recommandé d'utiliser la fonctionnalité intégrée de mise à jour de l'outil. Cette dernière est disponible par l'intermédiaire du menu Help/ Install New Software :



Dans l'écran correspondant, une liste de sites de mise à jour est proposée. Le site de Spring IDE 2.2 doit être alors ajouté par l'intermédiaire du bouton «Add». Entrez ensuite l'URL du site comme c'est indiqué dans la figure suivante :



Spring IDE offre une approche modulaire concernant ses différentes fonctionnalités. Il est ainsi possible de n'installer que les fonctionnalités souhaitées. Nous allons sélectionner tous les modules :

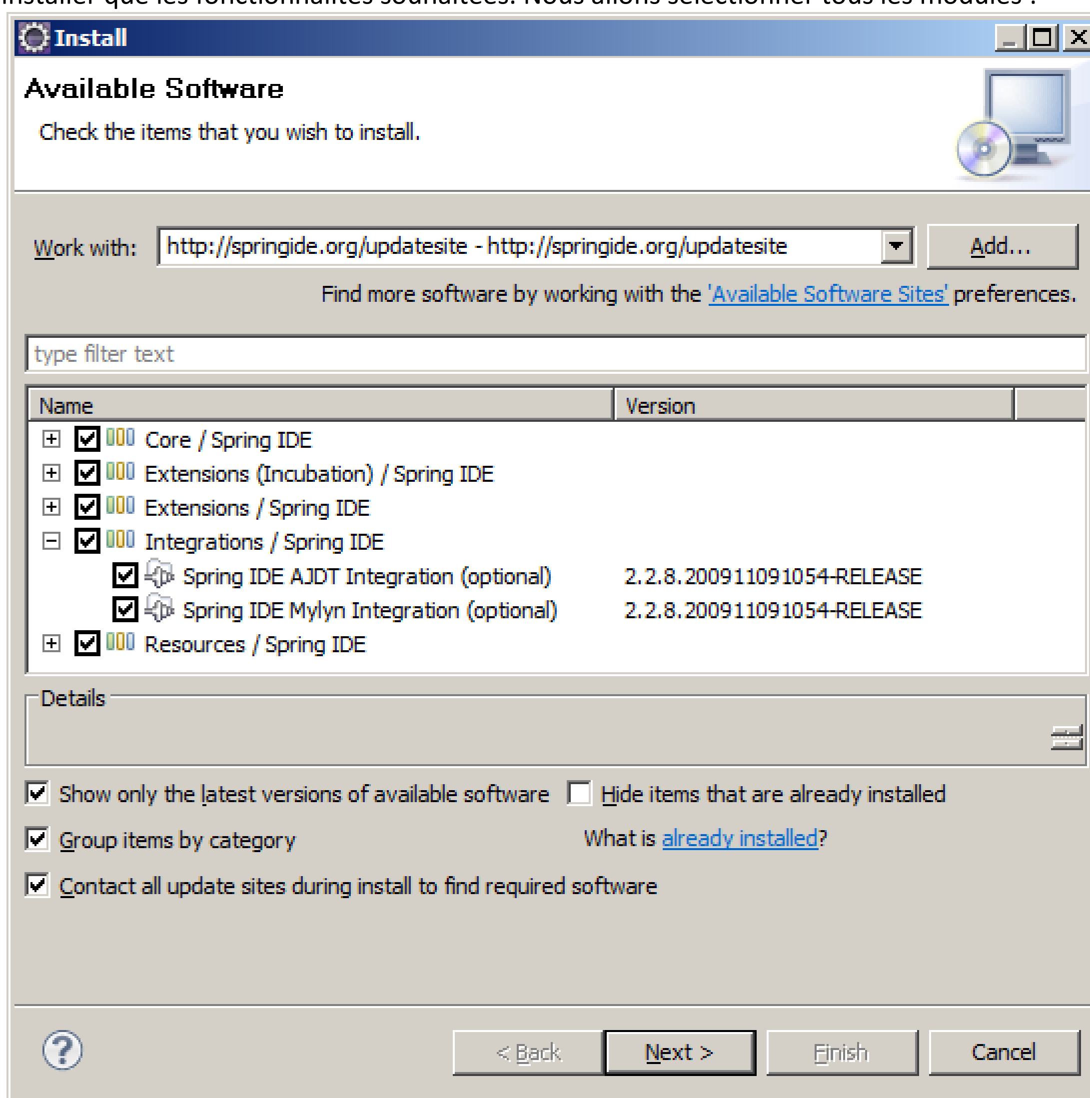


Tableau 0-1. Modules disponibles dans Spring IDE 2.2

Module	Groupe	Description
Spring IDE Core	Core	Core Correspond au cœur du greffon fournissant les différents outils relatifs afin de faciliter la mise en œuvre de Spring dans Eclipse.
Spring IDE AOP Extension	Extensions	Fournit un support pour l'espace de nommage AOP de Spring ainsi que la configuration d'aspects AspectJ par annotation.
Spring IDE OSGI Extension	Extensions	Fournit un support pour les développements utilisant Spring Dynamic Modules dans un environnement OSGi.
Spring IDE Security Extension	Extensions	Fournit un support pour Spring Security 2.0.
Spring IDE Web Flow Extension	Extensions	Met à disposition tous les outils afin de faciliter le développement d'applications Spring Web Flow dans Eclipse.

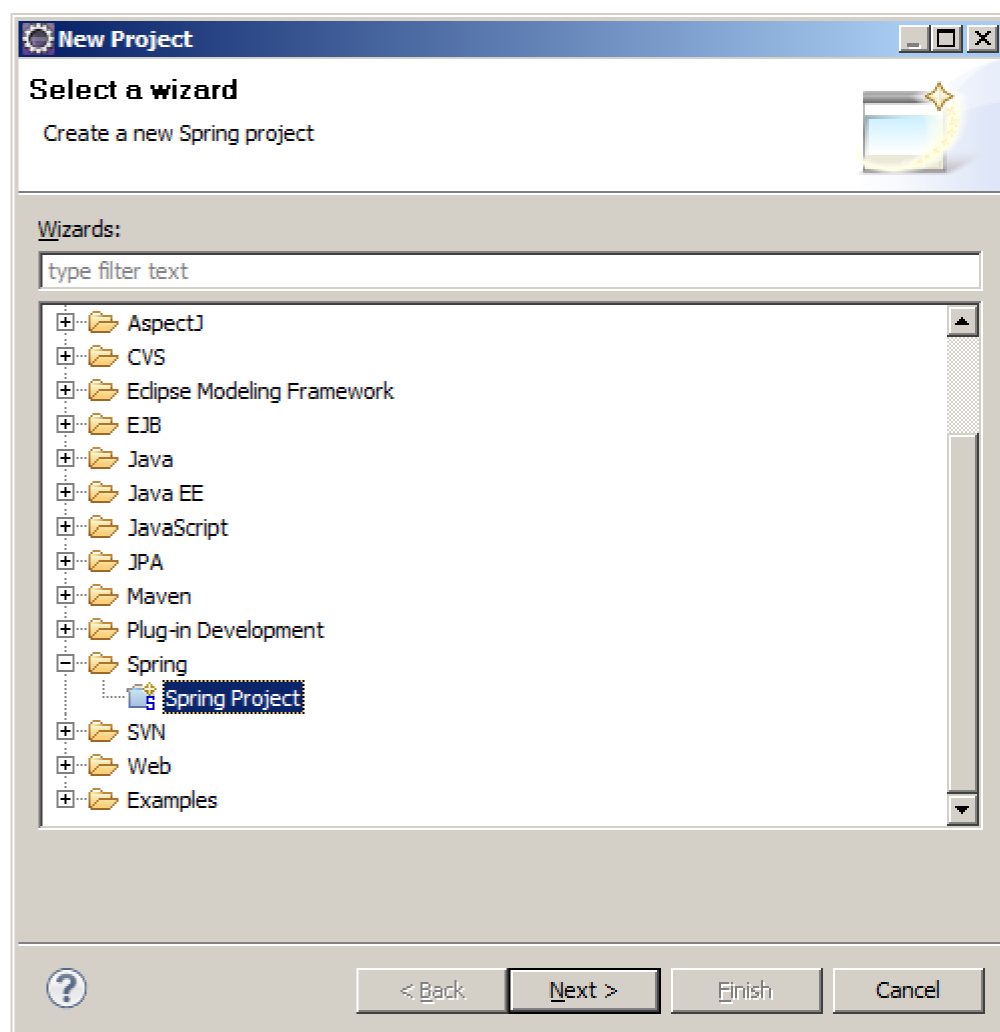
Spring IDE Autowire Extension	Extensions (Incubation)	Fournit un outil afin de supporter les configurations automatiques (autowiring) dans Spring.
Spring IDE JavaConfig Extension	Extensions (Incubation)	Fournit un support de l'outil JavaConfig de Spring.
Spring IDE AJDT Integration	Integrations	Met à disposition les briques relatives à l'intégration de Spring IDE avec AJDT.
Spring IDE Mylyn Integration	Integrations	Met à disposition les briques relatives à l'intégration de Spring

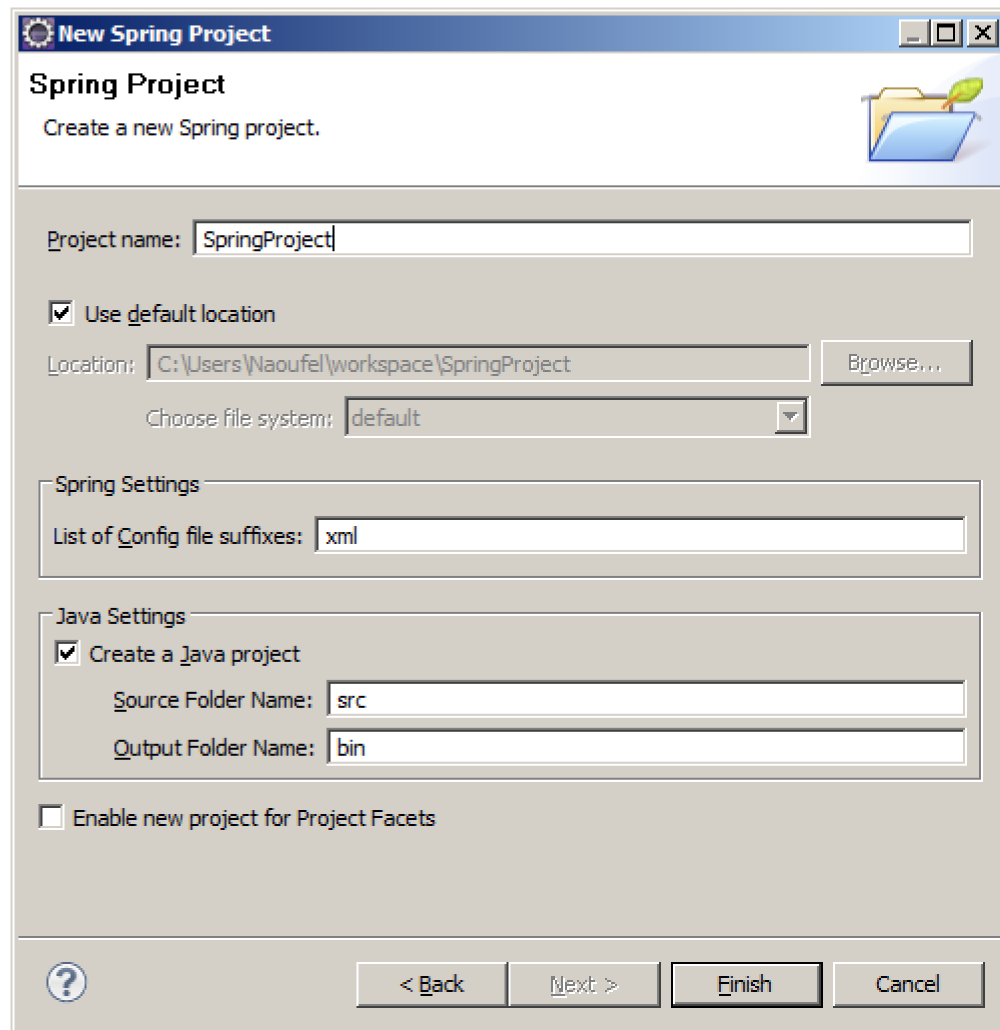
Acceptez ensuite les termes de la licence. Eclipse commence alors à télécharger les différents fichiers correspondant puis propose leur installation. Il convient alors de choisir le bouton « Install all » pour les installer. A la fin de l'installation, Eclipse proposera un redémarrage afin que les modules soient pris en compte. Une fois le redémarrage est terminé, les différents modules seront présents.

Projet de nature Spring

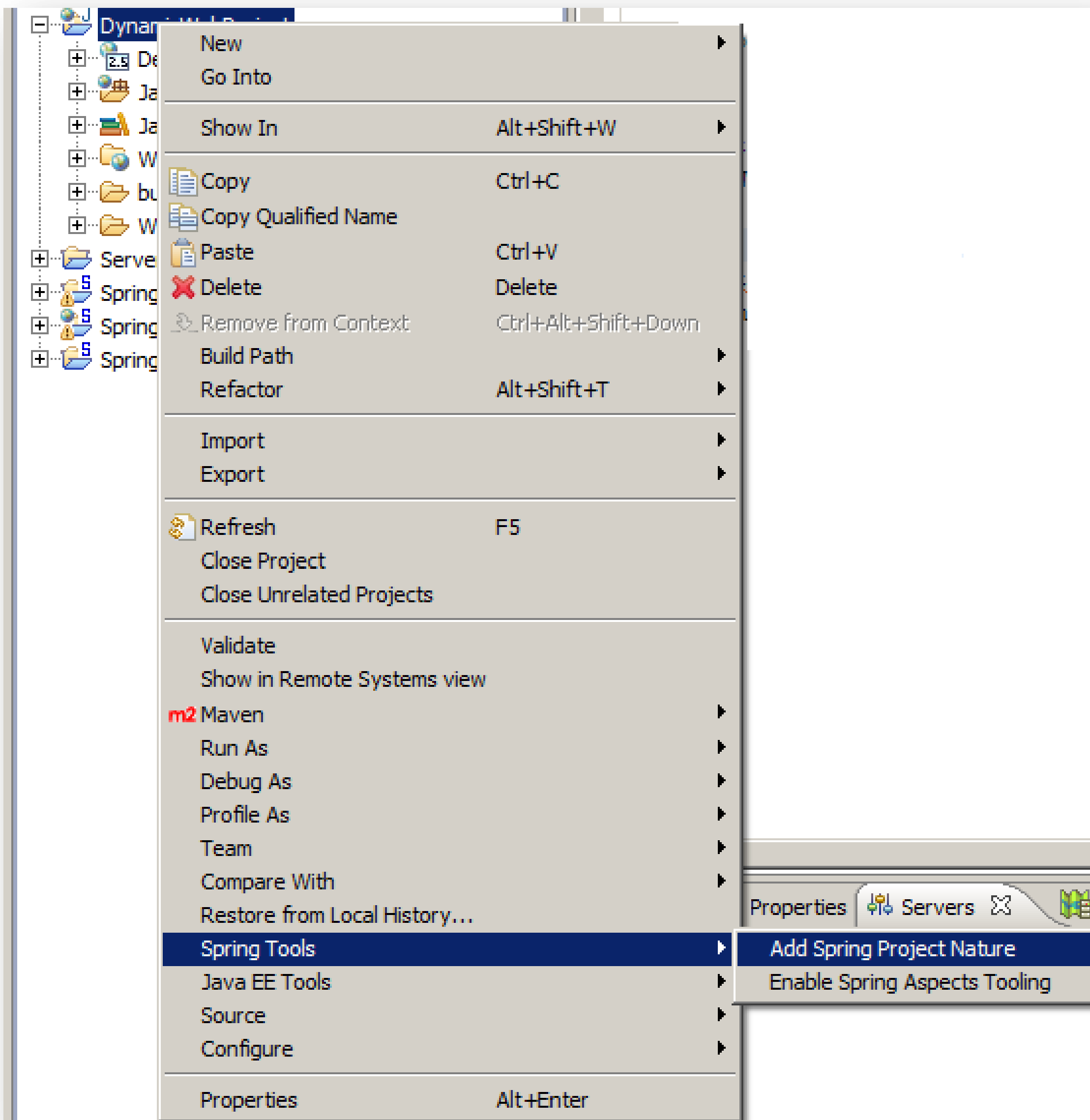
Spring IDE définit la notion de projet de nature Spring afin de marquer les projets utilisant Spring et contenant des fichiers de configuration XML relatifs.
Deux approches sont possibles afin de spécifier une nature Spring à un projet Java :

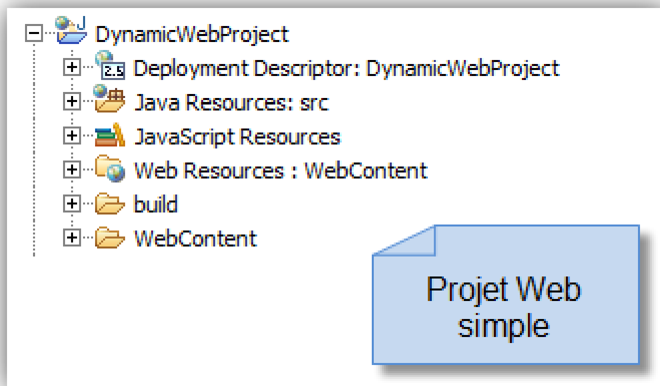
- La première peut être réalisé à la création du projet et consiste à la création d'un projet de type Spring par l'intermédiaire du menu contextuel « New/Other » puis la sélection de l'élément « Spring Project » sous la rubrique « Spring ».

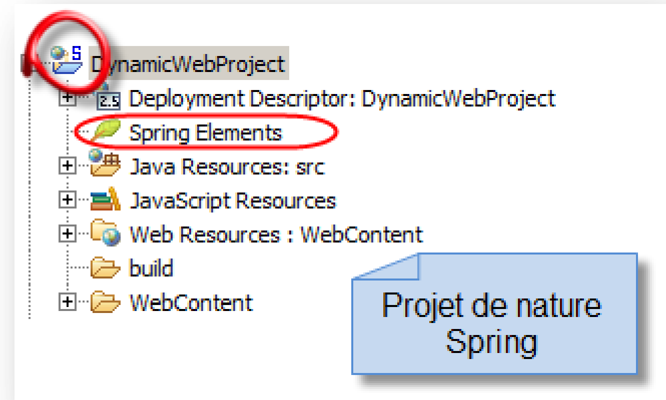



New Spring Project
Spring Project
Create a new Spring project.
Project name: SpringProject
Use default location
Location: C:\Users\Naoufel\workspace\SpringProject
Browse...
Choose file system: default
Spring Settings
List of Config file suffixes: xml
Java Settings
Create a Java project
Source Folder Name: src
Output Folder Name: bin
Enable new project for Project Facets
< Back
Next >
Finish
Cancel

- La nature Spring peut également être ajoutée manuellement par l'intermédiaire du menu contextuel accessible à partir du nom du projet. Il suffit de sélectionner l'élément « Add Spring Project Nature » dans la rubrique « Spring Tools » :

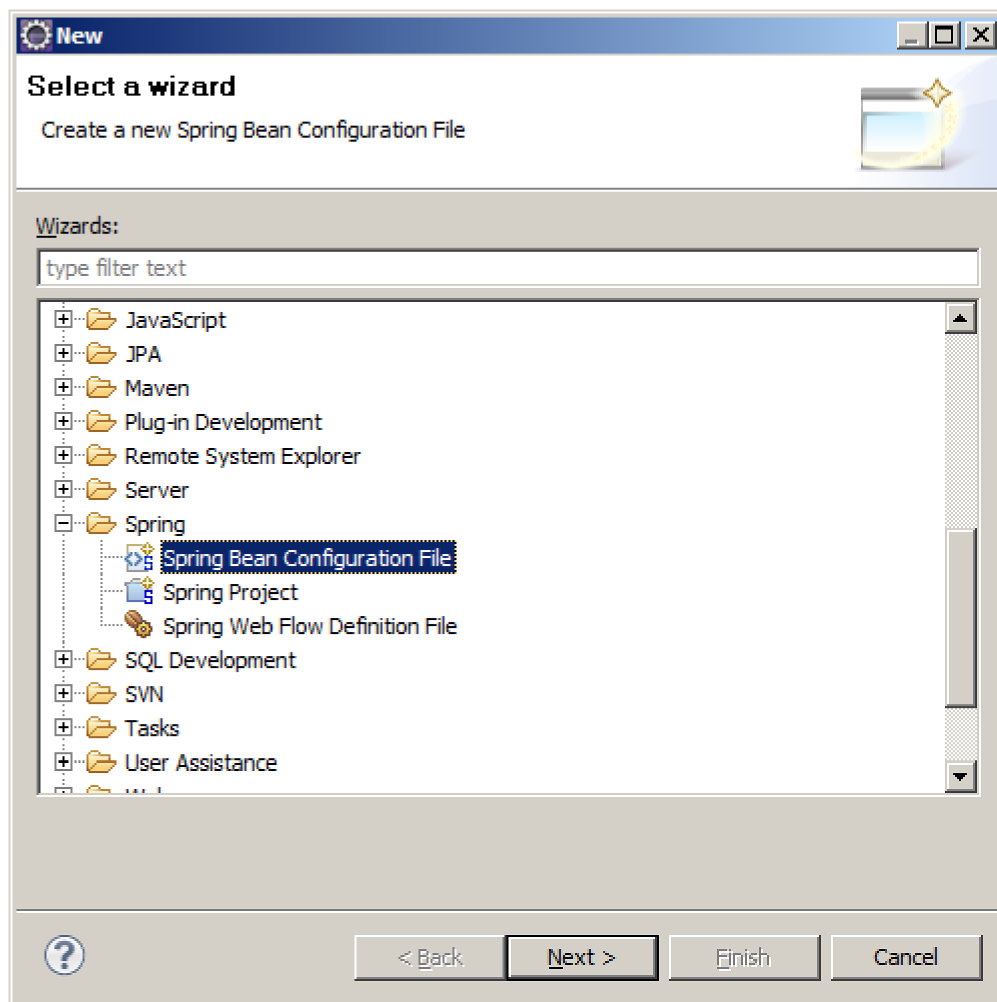




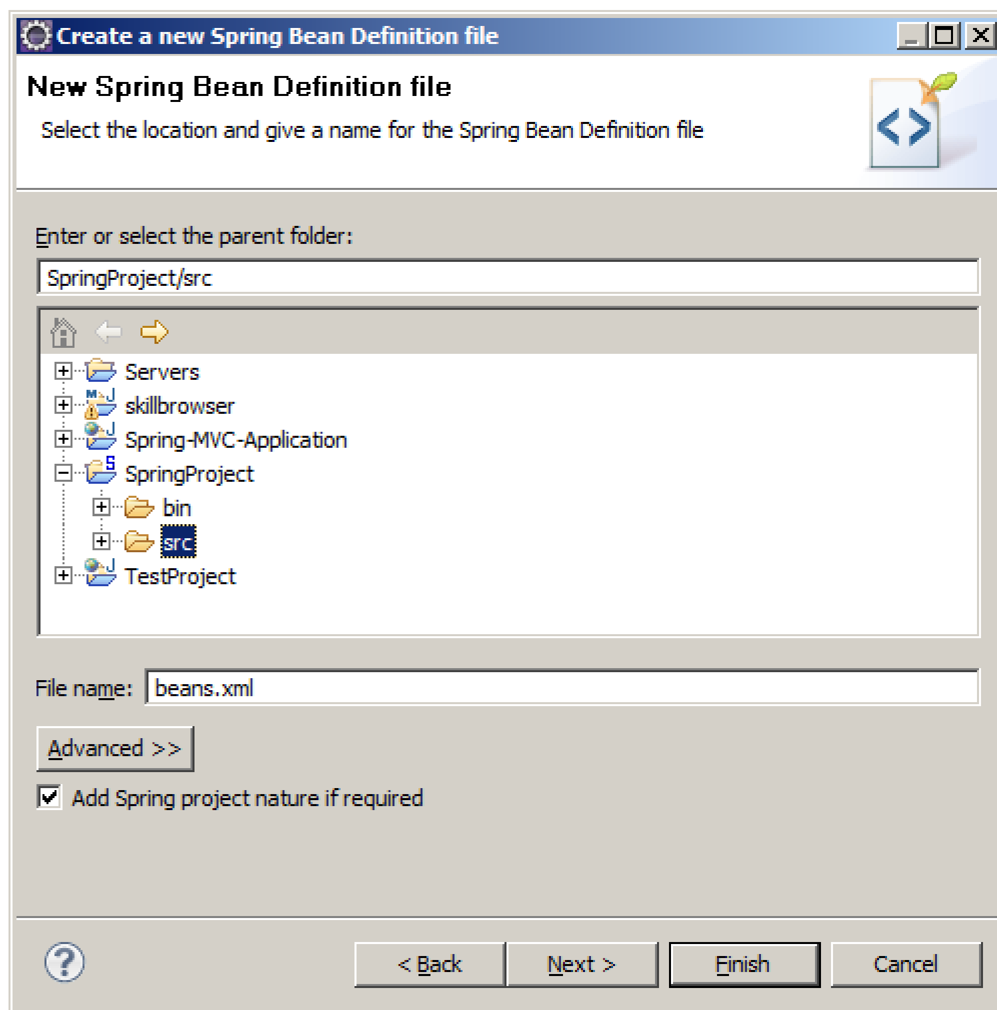


Fichier de configuration Spring

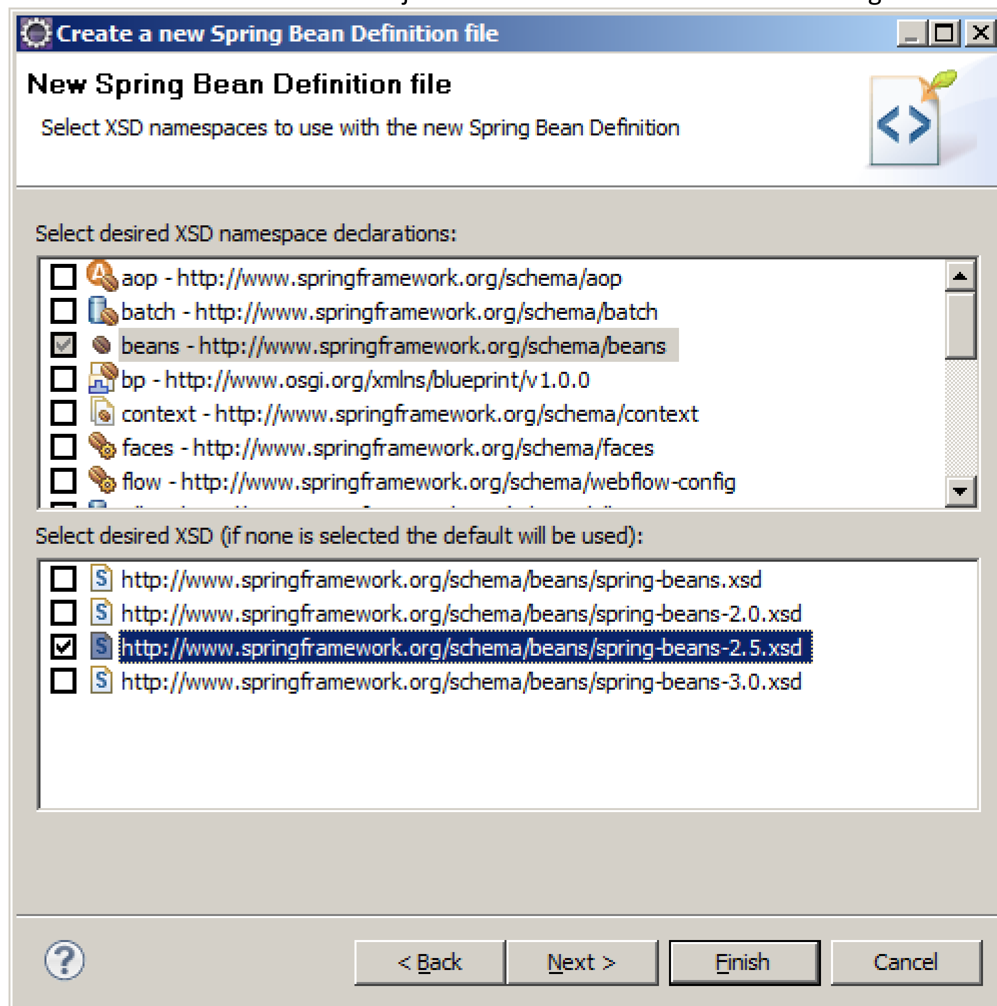
Il est à noter qu'il est également possible de créer un fichier de configuration Spring par l'intermédiaire du menu « New – Other » et choisir « Spring Bean Configuration File » sous la rubrique « Spring » :



Attribuez ensuite un nom au fichier de configuration :

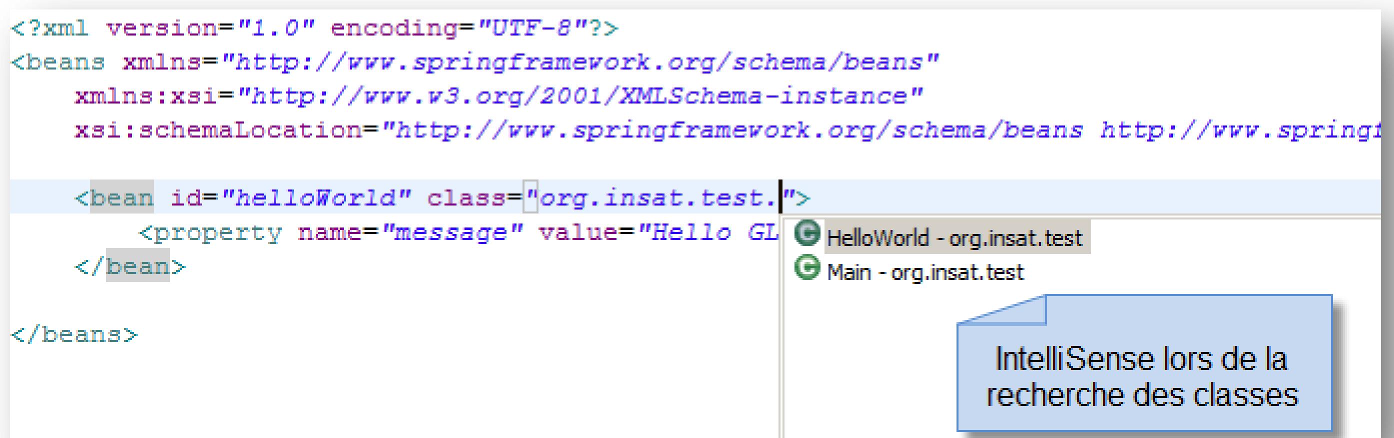


Cet élément permet de créer un squelette minimal pour un fichier de configuration Spring, à savoir la balise beans avec les différents espaces de nommage. Le wizard de création permet de sélectionner les espaces de nommage à inclure et si l'utilisateur souhaite ajouter le fichier à un ensemble de configuration :



Les principales facilités offertes dans l'édition d'un fichier XML de configuration de Spring sont les suivantes :

- Support relatif à la recherche des classes pour l'attribut class de la balise bean avec la complétion des éditeurs Eclipse :

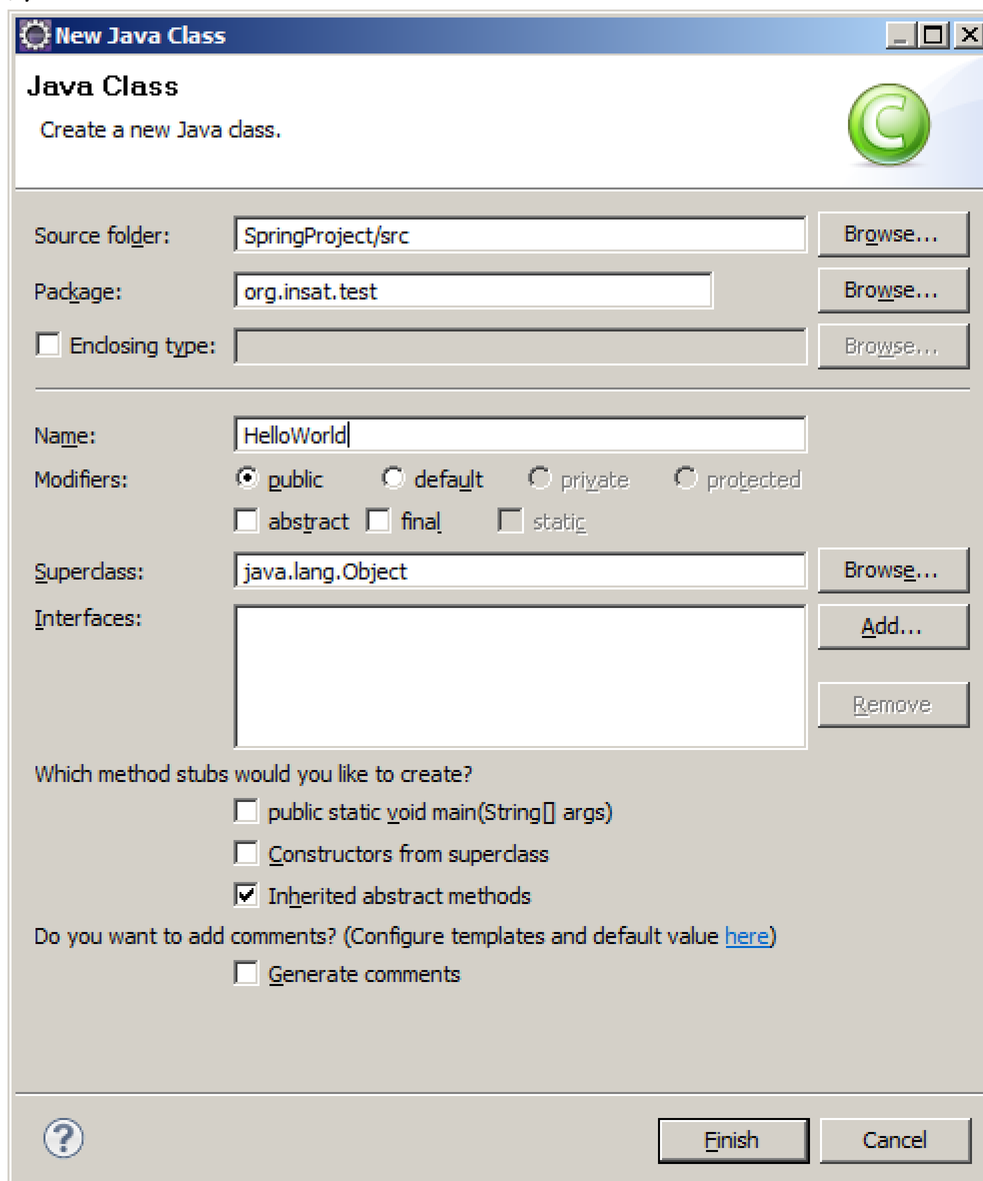


- Support relatif à la détection des propriétés utilisables pour un bean dans l'attribut name de la balise property avec la complétion ;
- Support relatif à la recherche des identifiants de beans pour un référencement de beans dans l'attribut ref de la balise property.

Ces fonctionnalités offrent un gain important de productivité lors lu développement d'applications Spring puisqu'elles permettent d'afficher les erreurs de la configuration XML avant de lancer l'application correspondante.

Application Spring HelloWorld

Créez un projet de nature Spring comme c'est déjà décrit.
Cliquez-droit sur le package « src » et créer un nouveau package « *org.insat.test* » et une classe « *HelloWorld* » :



Insérez le code suivant à la classe « HelloWorld » :

```
package org.insat.test;

public class HelloWorld {
	String message;

	public void setMessage(String message){
		this.message = message;
	}

	public void display(){
		System.out.println(message);
	}

}
```

La classe « HelloWorld » contient l'attribut « message » et la méthode setter « setMessage() ».
Au lieu d'attribuer la valeur du message directement, nous allons l'injecté à travers le fichier de configuration qu'on a déjà vu.
La classe HelloWorld dispose aussi de la méthode « *display()* » qui permet d'afficher le message.

Créez alors le fichier de configuration « bean configuration » nommé beans.xml comme c'est décrit dans la section « Fichier de configuration Spring ».

Insérez le code suivant au fichier de configuration beans.xml
.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

	<bean id="helloWorld" class="org.insat.test.HelloWorld">
		<property name="message" value="Hello GL INSAT"></property>
	</bean>

</beans>
```

L'attribut « i d » de l'élément « bean » est utilisé comme nom logique du bean et l'attribut « class » spécifie la classe correspondante.
L'élément « property » permet de donner la valeur « Hello GL INSAT » à l'attribut « message ».

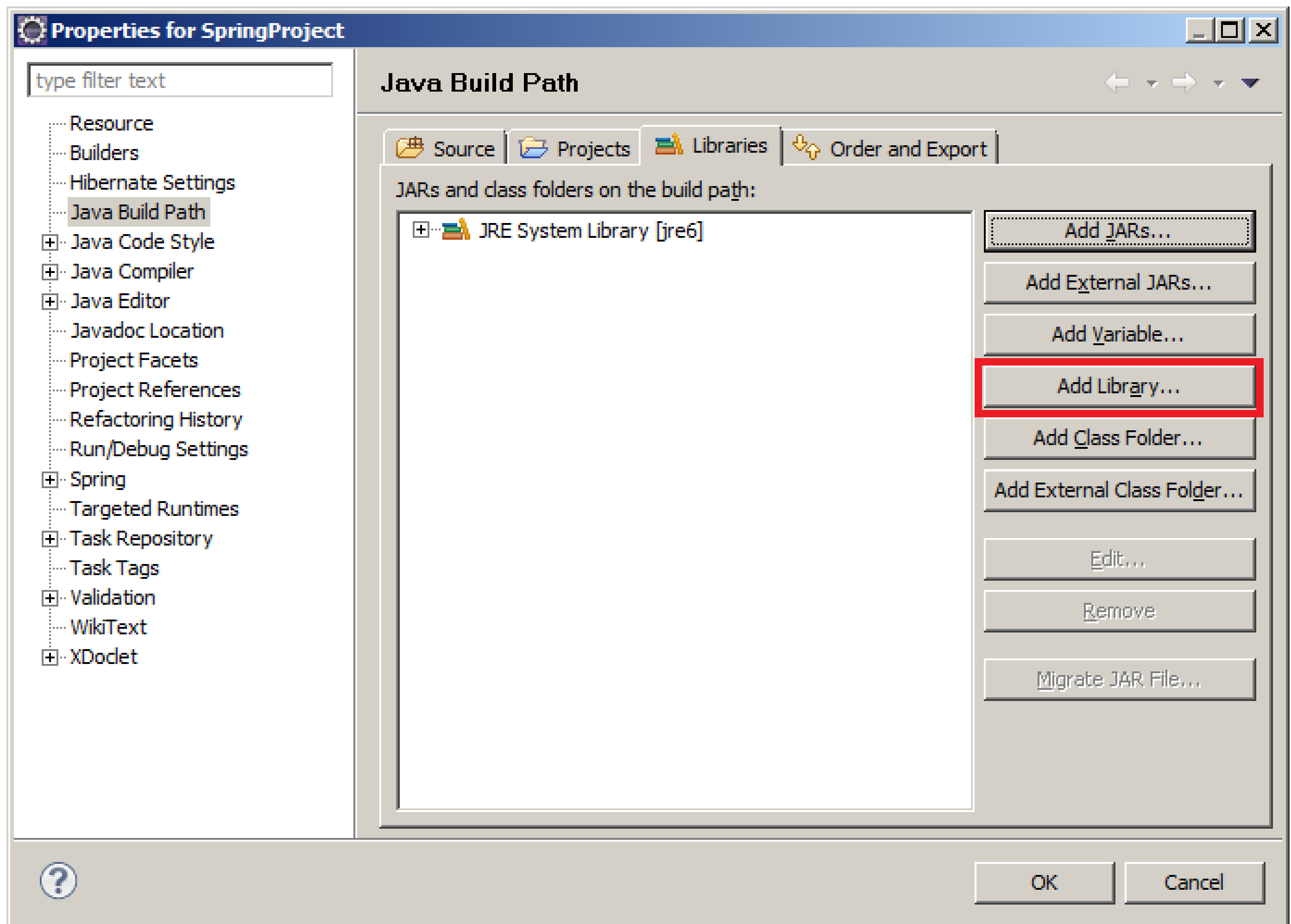
Ajout des librairies Spring

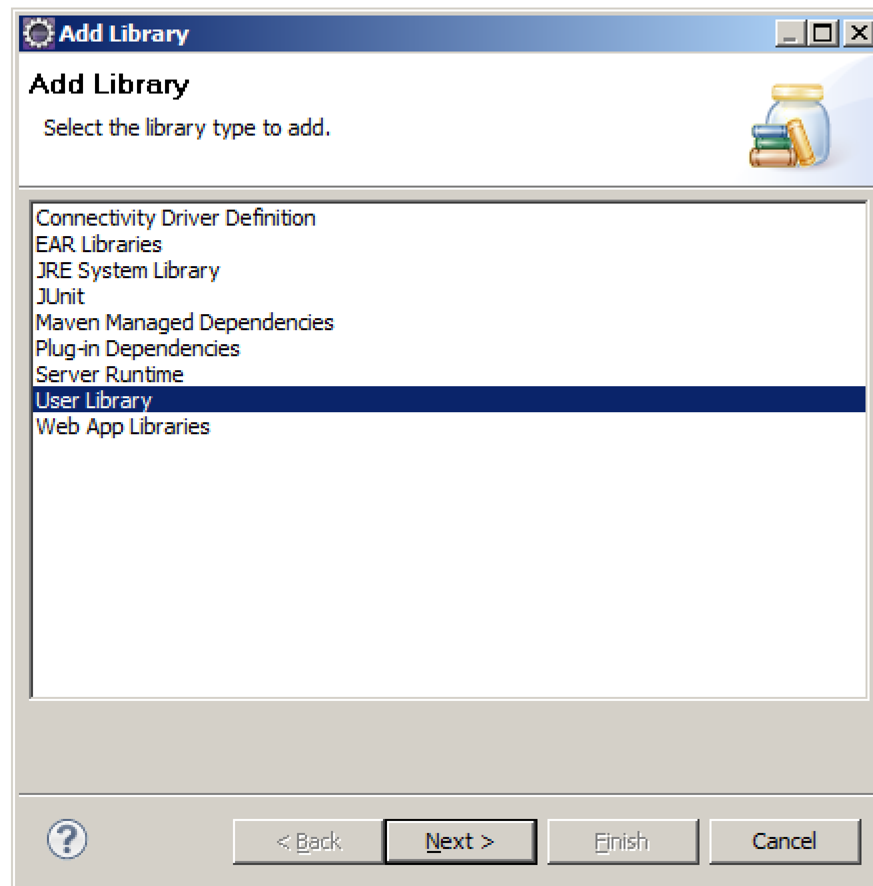
Pour exécuter le code, il nous faut les librairies suivantes :

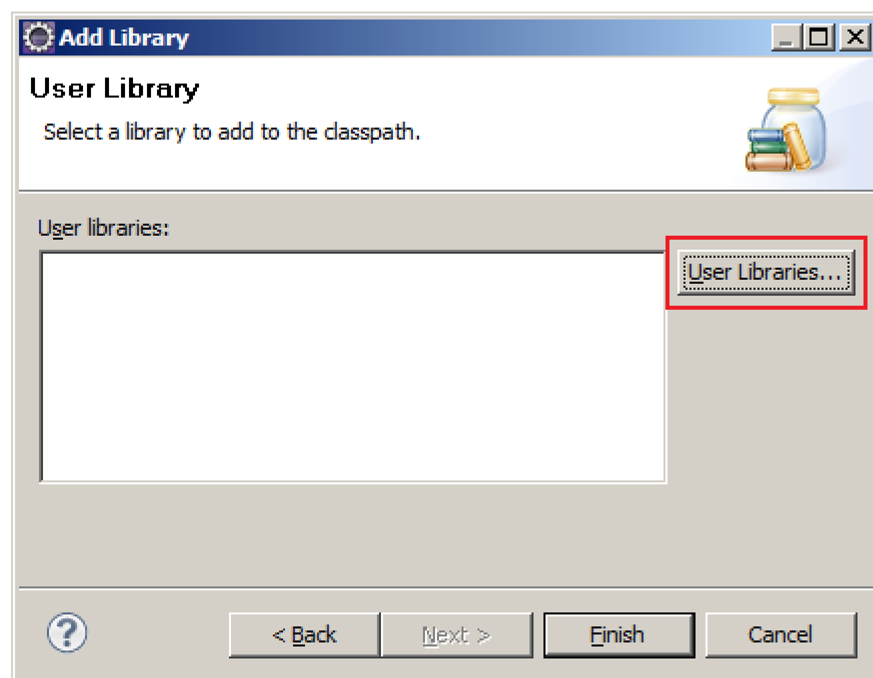
- Spring.jar (disponible sous ~\spring-framework-2.5.6.SEC01\dist)
- Commons-logging.jar (disponible sous ~\spring-framework-2.5.6.SEC01\lib)

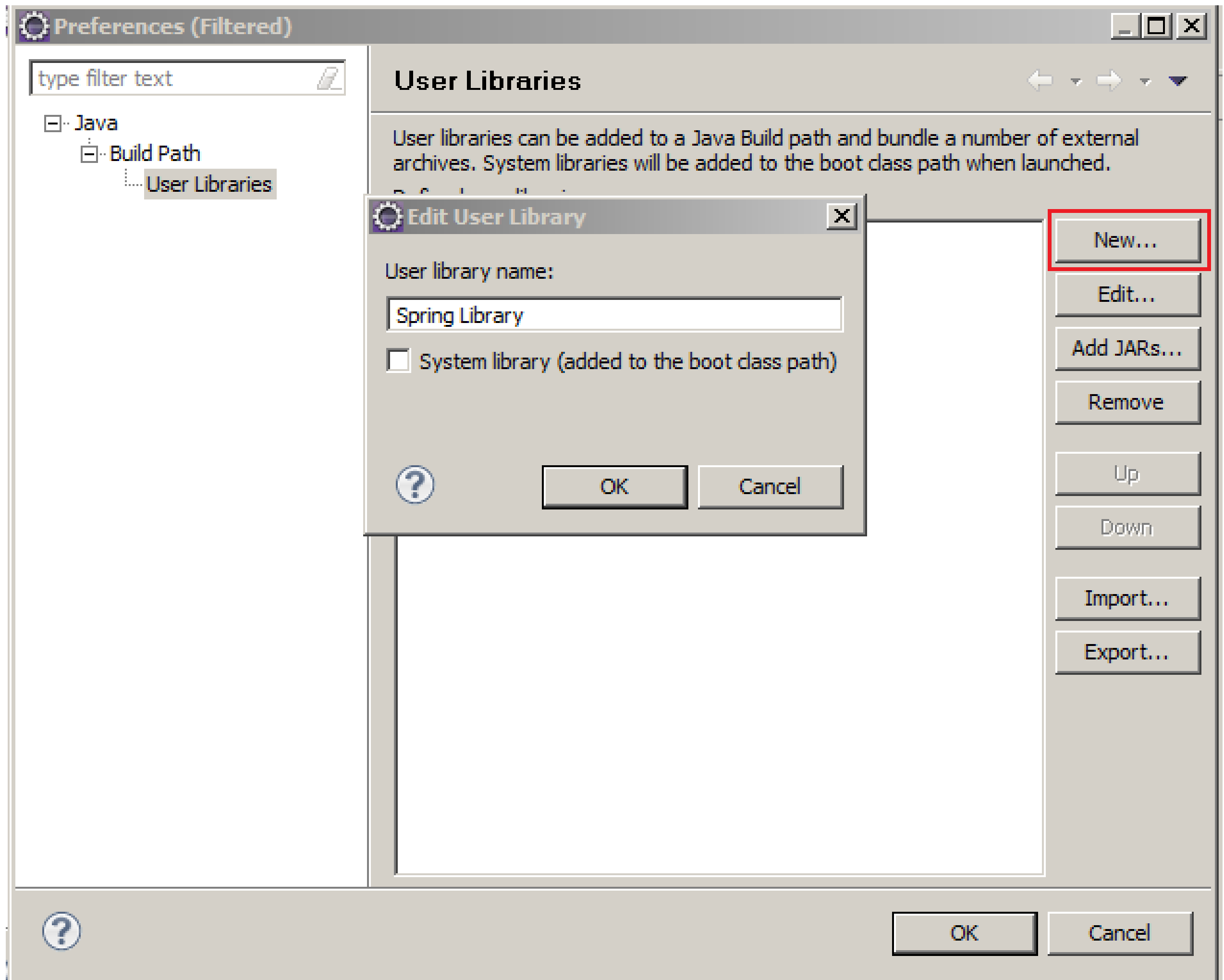
Pour ajoutez ces librairies ajoutez un dossier « lib »à la racine du projet et y mettre les jars nécessaires.

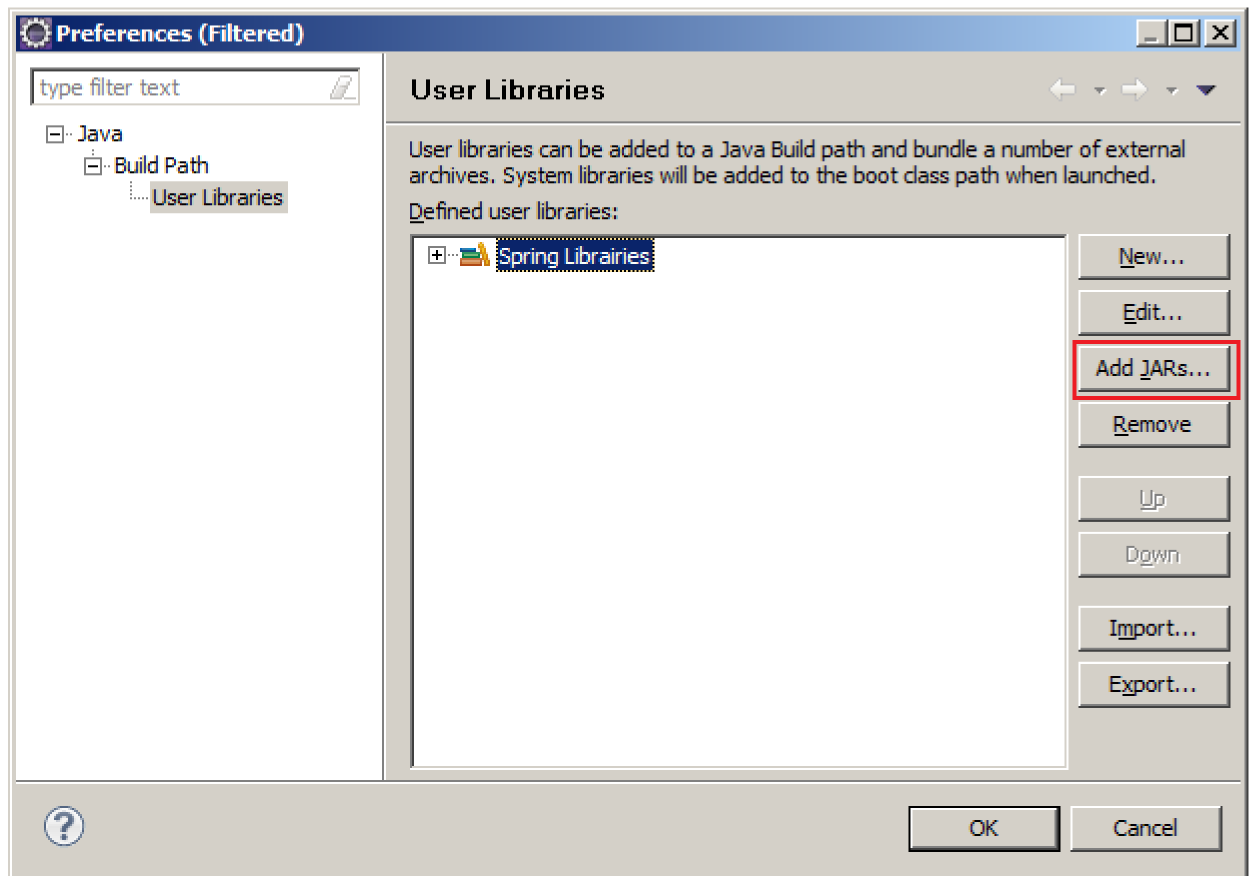
Cliquez-droit sur le projet puis aller vers « build path -> configure build path » et suivez les instructions suivantes :




Add Library
Add Library
Select the library type to add.
Connectivity Driver Definition
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries
< Back
Next >
Finish
Cancel


Add Library
User Library
Select a library to add to the classpath.
User libraries:
User Libraries...
< Back
Next >
Finish
Cancel


Preferences (Filtered)
type filter text
Java
Build Path
User Libraries
User Libraries
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.
Edit User Library
User library name:
Spring Library
System library (added to the boot class path)
OK
Cancel
New...
Edit...
Add JARs...
Remove
Up
Down
Import...
Export...
OK
Cancel



Sélectionnez enfin les 2 jars (spring.jar et commons-logging.jar).

Exécution du code

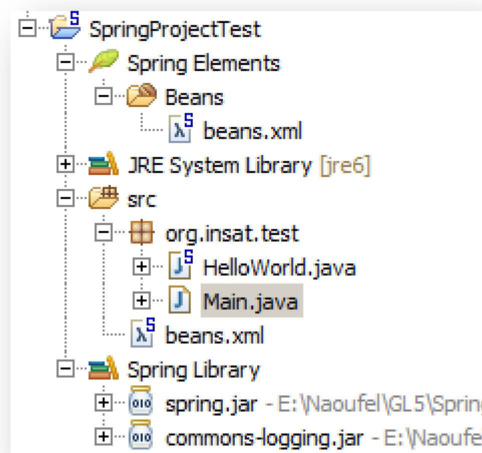
Nous allons maintenant afficher le message injecté à travers le fichier de configuration.

Créez la classe « Main » et y insérer le code suivant :

```
package org.insat.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

	public static void main(String[] args) {
		ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
		HelloWorld helloWorld = (HelloWorld) context.getBean("helloWorld");
		helloWorld.display();

	}
}
```

Nous avons instancié le « IoC container » avec le fichier de configuration beans.xml. La méthode « getBean() » permet de localiser la classe helloWorld. Enfin la méthode « display() » permet d'afficher le message.

Arborescence finale du projet



Refactoring

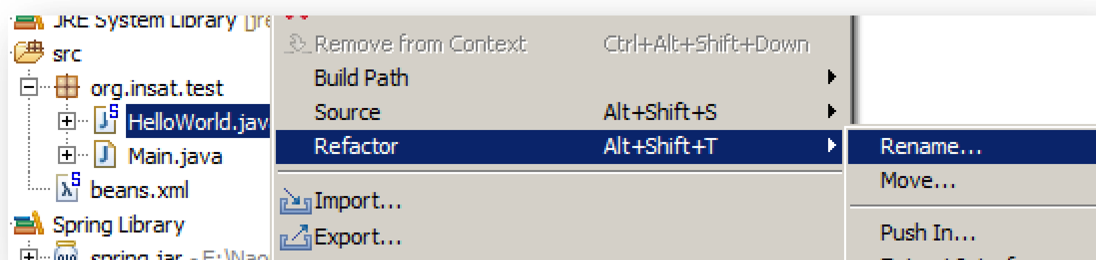
Spring IDE s'intègre parfaitement dans les mécanismes de « refactoring » mis à disposition par l'environnement de développement Eclipse

Cet aspect offre notamment la possibilité de garder une synchronisation entre les entités et leurs configurations relatives dans Spring à la suite d'un renommage ou un déplacement.

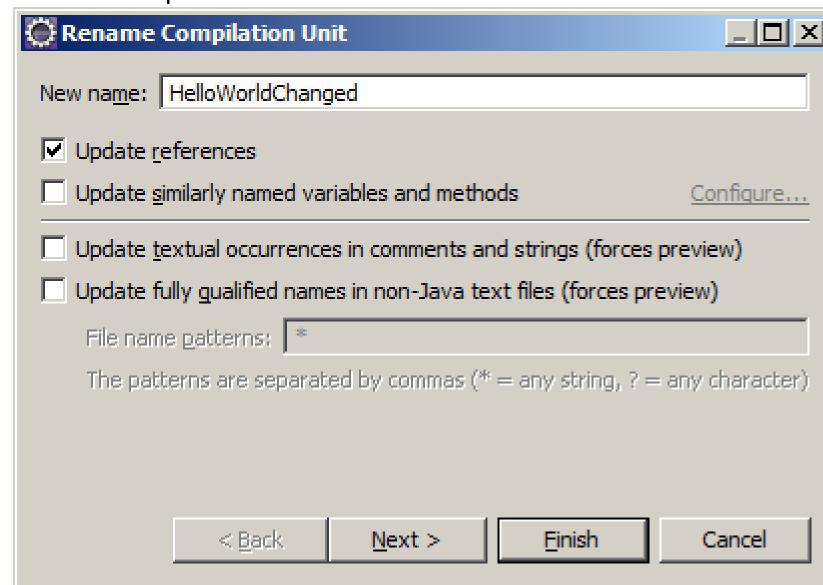
Il permet aussi bien de visualiser les impacts sur les fichiers de configuration de Spring.

Nous allons tester cet aspect en changeant le nom de la classe « HelloWorld ».

Cliquez-droit sur la classe « HelloWorld » puis « Refactor -> Rename » :



Changez le nom de la classe et cliquez sur « **Next** »:



Rename Compilation Unit

Changes to be performed

Main.java - SpringProjectTest/src/org/insat/test

Rename compilation unit 'HelloWorld.java' to 'HelloWorldChanged.java'

Rename classes referenced in Spring Bean definitions

beans.xml - SpringProjectTest/src

beans.xml

Original Source	Refactored Source
.org/schema/beans"	.org/schema/beans"
LSchema-instance"	LSchema-instance"
gframework.org/schema/beans http:/	gframework.org/schema/beans htt
at.test.HelloWorld">	at.test.HelloWorldChanged">
Hello GL INSAT"></property>	Hello GL INSAT"></property>

< Back Next > Finish Cancel

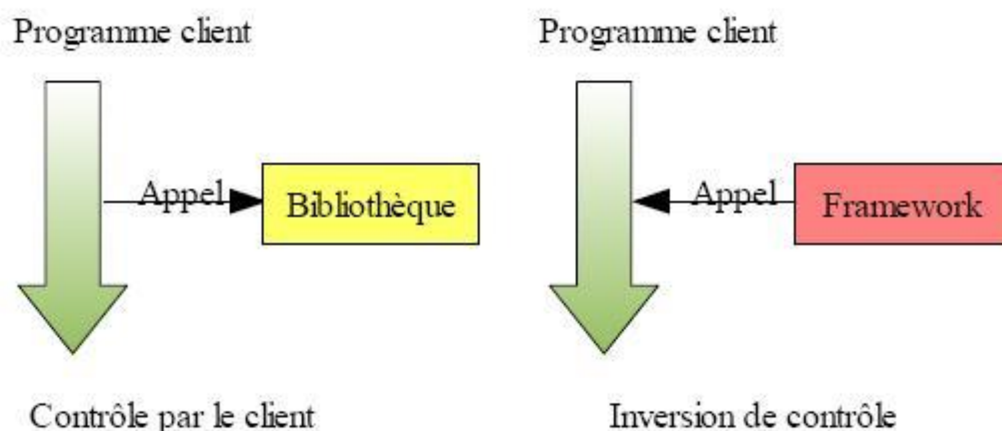
Atelier 2: Inversion of Contrôle (IoC)

Atelier 2 - Inversion of Contrôle (IoC)

Principe

L'inversion de contrôle est un motif de conception logicielle (***design pattern***) commun à tous les frameworks, devenu populaire avec l'adoption des conteneurs dits "légers". Cette notion fait partie des principes de la programmation orientée aspect.

Selon le mécanisme du *design pattern, c*e n'est plus l'application qui appelle les fonctions d'une librairie, mais un framework qui, à l'aide d'une couche abstraite mettant en œuvre un comportement propre, va appeler l'application en l'implémentant. L'inversion de contrôle s'utilise par héritage de classes du framework ou par le biais d'un mécanisme de plug-in.



L'idée principale est que les modules de hauts et bas niveaux doivent dépendre d'interfaces, et non les modules dépendre les uns des autres. Les modules dérivent des interfaces. L'inversion de contrôle les rend donc indépendants entre eux. L'utilisation d'une interface permet de s'abstraire de la source de données. Ainsi les composants ou les services, qui sont des composants distants, forment des entités autonomes et réutilisables sans avoir modifier l'ensemble des codes sources.

Ce motif de conception découpe l'application en plusieurs blocs de code indépendants, pris en charge par l'API fournie avec le framework. L'application gère l'IHM et le traitement, et le framework prend en charge l'exécution de ce traitement. Au lieu d'appeller une classe pour exécuter une action, un plug-in de n'importe quel utilisateur injecte l'implémentation dans l'instanciation. Ce modèle autorise plusieurs implémentations de classes différentes.

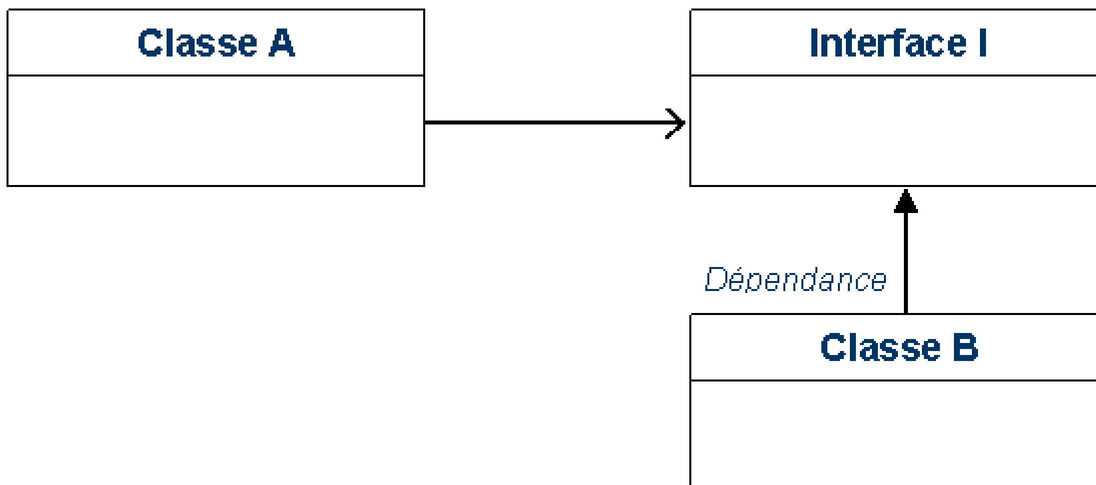
L'inversion de contrôle la plus connue est l'injection de dépendances (**Dependency Injection**). Il existe trois types d'injection de dépendances :

- **L'injection par constructeurs :** Ce type d'injection se fait sur le constructeur, c'est-à-dire que le constructeur dispose de paramètres pour directement initialiser tous les membres de la classe.
- **L'injection par mutateurs (setters) :** Ce type d'injection se fait après une initialisation à l'aide d'un constructeur sans paramètre puis les différents champs sont initialisés grâce à des mutateurs.
- **L'injection d'interface :** Cette injection se fait sur la base d'une méthode, elle est plus proche de l'injection par mutateurs, enfin la différence se résume à pouvoir utiliser un autre nom de méthode que ceux du "design pattern bean". Pour cela, il faut utiliser une interface afin de définir le nom de la méthode qui injectera la dépendance.

Injection de dépendances :

A dépend directement de **B**

A dépend de l'interface **I**, et **B** est une implémentation de l'interface **I** => Diminuer le couplage



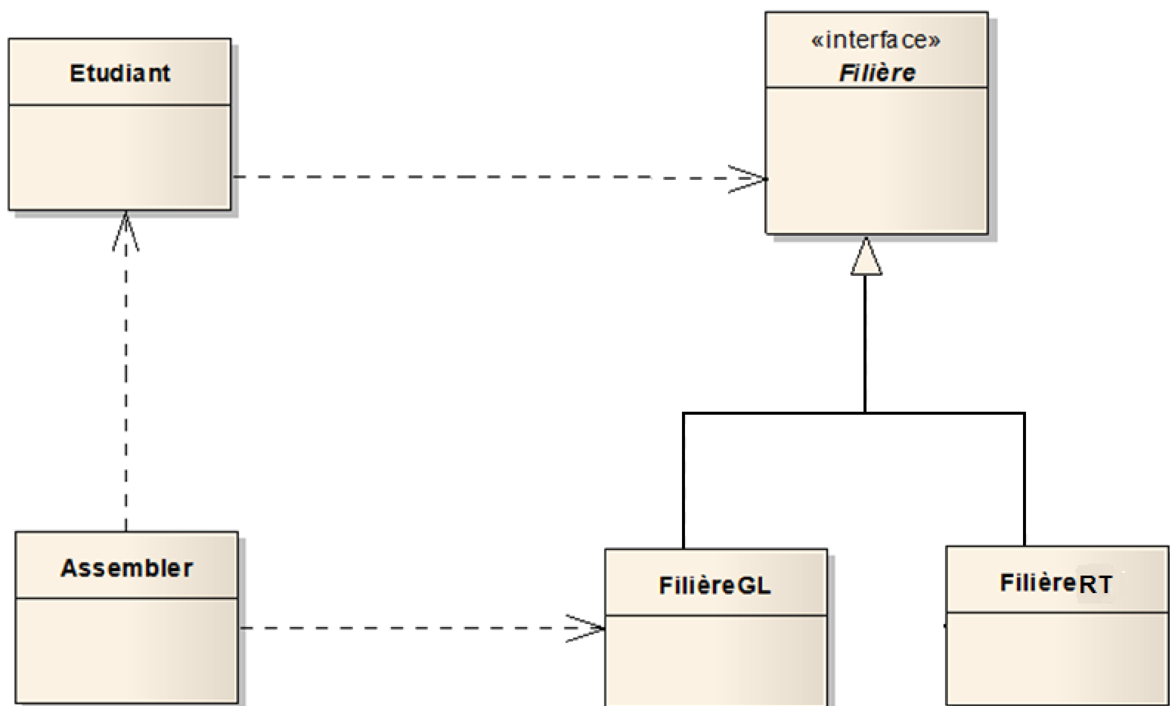
Exemple d'utilisation

Dans cet atelier, nous allons appliquer l'inversion de contrôle en utilisant le framework spring.

- Créer un 'projet Spring' à l'aide du plugin eclipse Spring IDE (voir Atelier Spring IDE).
- Ajouter les jars suivants à votre application :
  - **spring.jar** présent dans \SPRING_HOME\dist
  - **commons-logging.jar** présent dans \SPRING_HOME\ lib\jakarta-commons
  - **aspectjweaver.jar** présent dans \SPRING_HOME\ lib\aspectj
  - **aspectjrt.jar** présent dans \SPRING_HOME\ lib\aspectj
  - **cglib-nodep-2.1_3.jar** présent dans \SPRING_HOME\ lib\cglib

Avec « SPRING_HOME » désigne le chemin d'installation du framework spring.

Soit l'exemple suivant : un étudiant est caractérisé par sa filière. Donc soit la classe **Etudiant** contenant un attribut **Filiere**. **Filiere** est une interface implémentée par deux classes : **FiliereGL** et **FiliereRT**. Ci-dessous le diagramme de classe correspondant à cet exemple :



Voici la classe Etudiant :

```
package org.insat.ioc;

public class Etudiant {

	private Filiere filiere;

	public Etudiant() {
	}

	public Etudiant(Filiere filiere) {
		this.filiere = filiere;
	}

	public void setFiliere(Filiere filiere) {
		this.filiere = filiere;
	}

	public Filiere getFiliere() {
		return filiere;
	}

	public void maFiliere(){
		filiere.afficherFiliere();
	}
}
```

Voici le code de l'interface **Filiere** et ses deux implémentations **FiliereGL** et **FiliereRT**:

```
package org.insat.ioc;

public interface Filiere {

	public void afficherFiliere();

}
```

```
package org.insat.ioc;

public class FiliereGL implements Filiere {

	@Override
	public void afficherFiliere() {
		System.out.println("Je suis un GLien :) Vive GL!");
	}
}
```

```java
package org.insat.ioc;

public class FiliereRT implements Filiere {

	@Override
	public void afficherFiliere() {
		System.out.println("Je suis un RTien :) Vive RT!");
	}
}
```

Nous allons déclarer l'instanciation de la classe **Etudiant** et lui attribuer une instance d'une implémentation de **Filiere**, dans le fichier de définition des beans « **beans.xml** » de Spring. Nous allons le faire par ces différentes méthodes :

L'injection par mutateurs (setters) :

Dans ce qui suit, nous allons modifier la référence de l'objet **Filiere** de la classe **Etudiant** en utilisant une injection par mutateur, le contenu de beans.xml est le suivant :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

	<bean id="filiereGL" class="org.insat.ioc.FiliereGL" />
	<bean id="filiereRT" class="org.insat.ioc.FiliereRT" />
	<bean id="etudiant" class="org.insat.ioc.Etudiant">
		<property name="filiere">
			<ref local="filiereGL" />
		</property>
	</bean>

</beans>
```

L'injection par constructeur:

Dans ce qui suit, nous allons modifier la référence de l'objet **Filiere** de la classe **Etudiant** en utilisant une injection par constructeur, le contenu de beans.xml est le suivant :

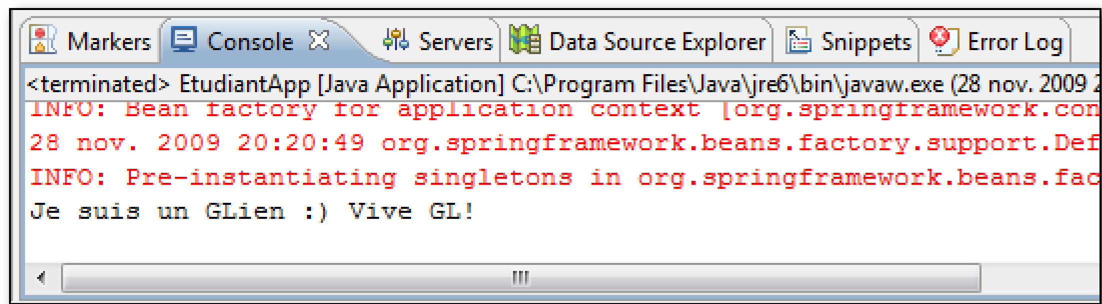
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">


      <bean id="filiereGL" class="org.insat.ioc.FiliereGL" />
      <bean id="filiereRT" class="org.insat.ioc.FiliereRT" />
      <bean id="etudiant" class="org.insat.ioc.Etudiant">
            <constructor-arg index="0" type="org.insat.ioc.Filiere" >
                  <ref local="filiereGL"/>
            </constructor-arg>
      </bean>

</beans>
```

On ajoute l'attribut **index** dans la balise **constructor-arg** pour identifier l'ordre de cet argument dans la liste des arguments du constructeur, et l'attribut **type** pour expliciter le type de l'argument (utile dans le cas de surcharge de constructeur). C'est deux attributs, **index** et **type**, sont facultatifs et utilisés lorsqu'il y a d'ambiguïté.

Le programme principal est le suivant :

```
package org.insat.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class EtudiantApp {

      public static void main(String[] args) {
            ApplicationContext context = new
                              ClassPathXmlApplicationContext("beans.xml");
            Etudiant etudiant = (Etudiant)context.getBean("etudiant");
            etudiant.maFiliere();
      }

}
```

L'exécution de ce programme:



Amélioration

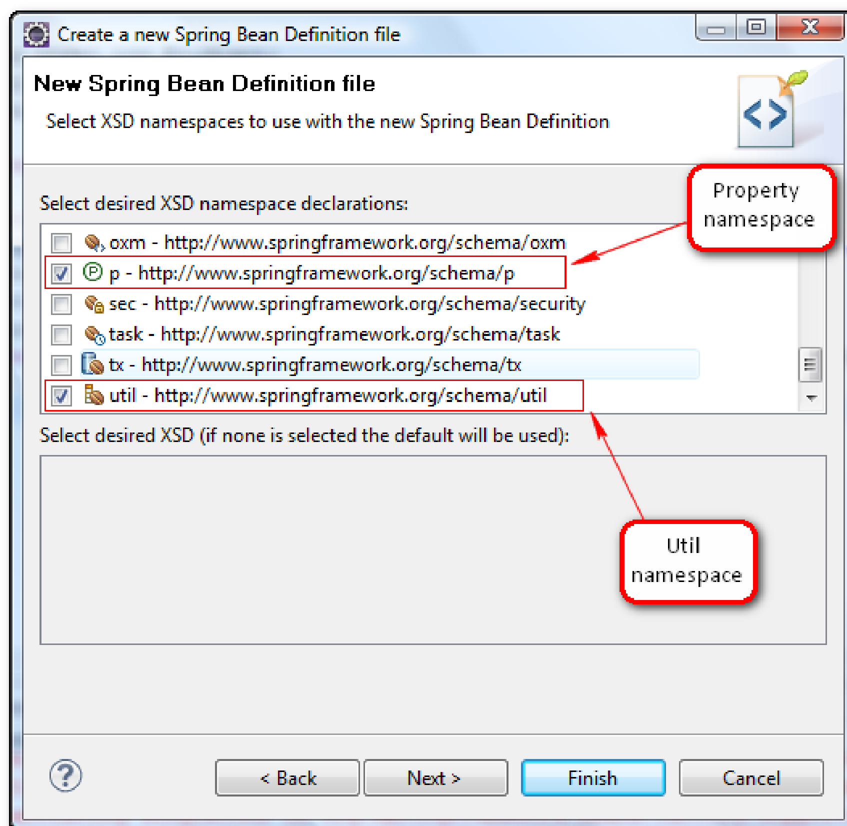
Si nous voulons améliorer notre programme par l'ajout d'un Map contenant la liste des matières et leurs coefficients d'un étudiant, en ajoutant l'attribut **matieres** dans la classe **Etudiant** comme suit :

```
	private Map<String, Double> matieres;

	public Map<String, Double> getMatieres() {
		return matieres;
	}

	public void setMatieres(Map<String, Double> matieres) {
		this.matieres = matieres;
	}
```

Le fichier de définition des beans « **beans.xml** » sera comme suit (utilisation de la méthode : injection par mutateur) :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <bean id="filiereGL" class="org.insat.ioc.FiliereGL" />
      <bean id="filiereRT" class="org.insat.ioc.FiliereRT" />
      <bean id="etudiant" class="org.insat.ioc.Etudiant">
            <property name="filiere">
                  <ref local="filiereGL" />
            </property>
            <property name="matieres">
                  <map>
                        <entry key="Programmation" value="3" />
                        <entry key="Architecture" value="2" />
                        <entry key="SIG" value="1.5" />
                        <entry key="Anglais" value="1" />
                  </map>
            </property>
      </bean>

</beans>
```

On peut améliorer aussi notre fichier de définition des beans « **beans.xml** », en utilisant les namespaces **property** et **util**, lors de la création de ce dernier :



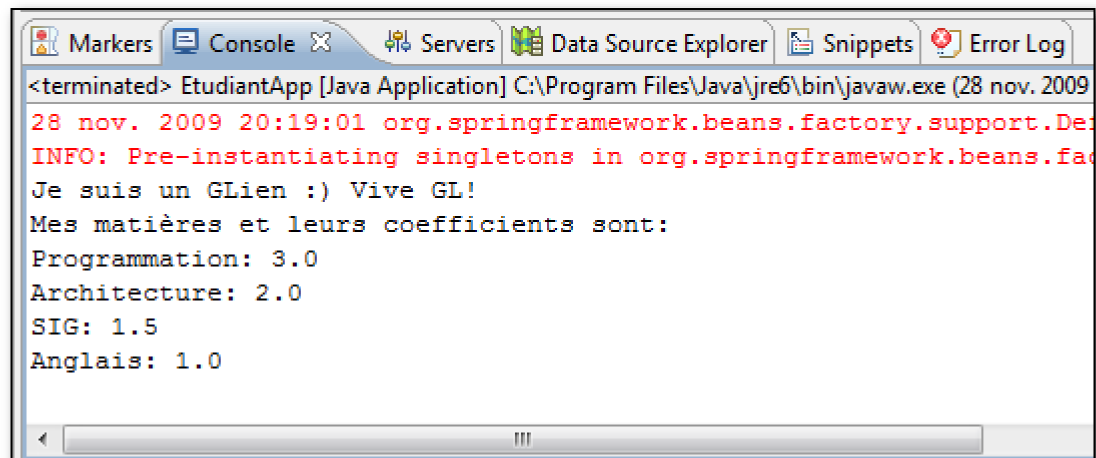
Et le fichier de définition des beans « **beans.xml** » sera :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.5.xsd">


      <bean id="filiereGL" class="org.insat.ioc.FiliereGL" />
      <bean id="filiereRT" class="org.insat.ioc.FiliereRT" />

      <util:map id="matieres">
            <entry key="Programmation" value="3" />
            <entry key="Architecture" value="2" />
            <entry key="SIG" value="1.5" />
            <entry key="Anglais" value="1" />
      </util:map>

      <bean id="etudiant" class="org.insat.ioc.Etudiant" p:filiere-ref="filiereGL"
p:matieres-ref="matieres" />

</beans>
```

Et le programme principale sera le suivant :

```
package org.insat.main;

import java.util.Map;

import org.insat.ioc.Etudiant;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class EtudiantApp {

      public static void main(String[] args) {
            ApplicationContext context = new
                        ClassPathXmlApplicationContext("beans.xml");
            Etudiant etudiant = (Etudiant)context.getBean("etudiant");
            etudiant.maFiliere();
            System.out.println("Mes matières et leurs coefficients sont: ");
            for (Map.Entry<String, Double> entry :
                              etudiant.getMatieres().entrySet()) {
                  System.out.println(entry.getKey() + ": " + entry.getValue());
            }
      }
}
```

L'exécution de ce programme résulte :



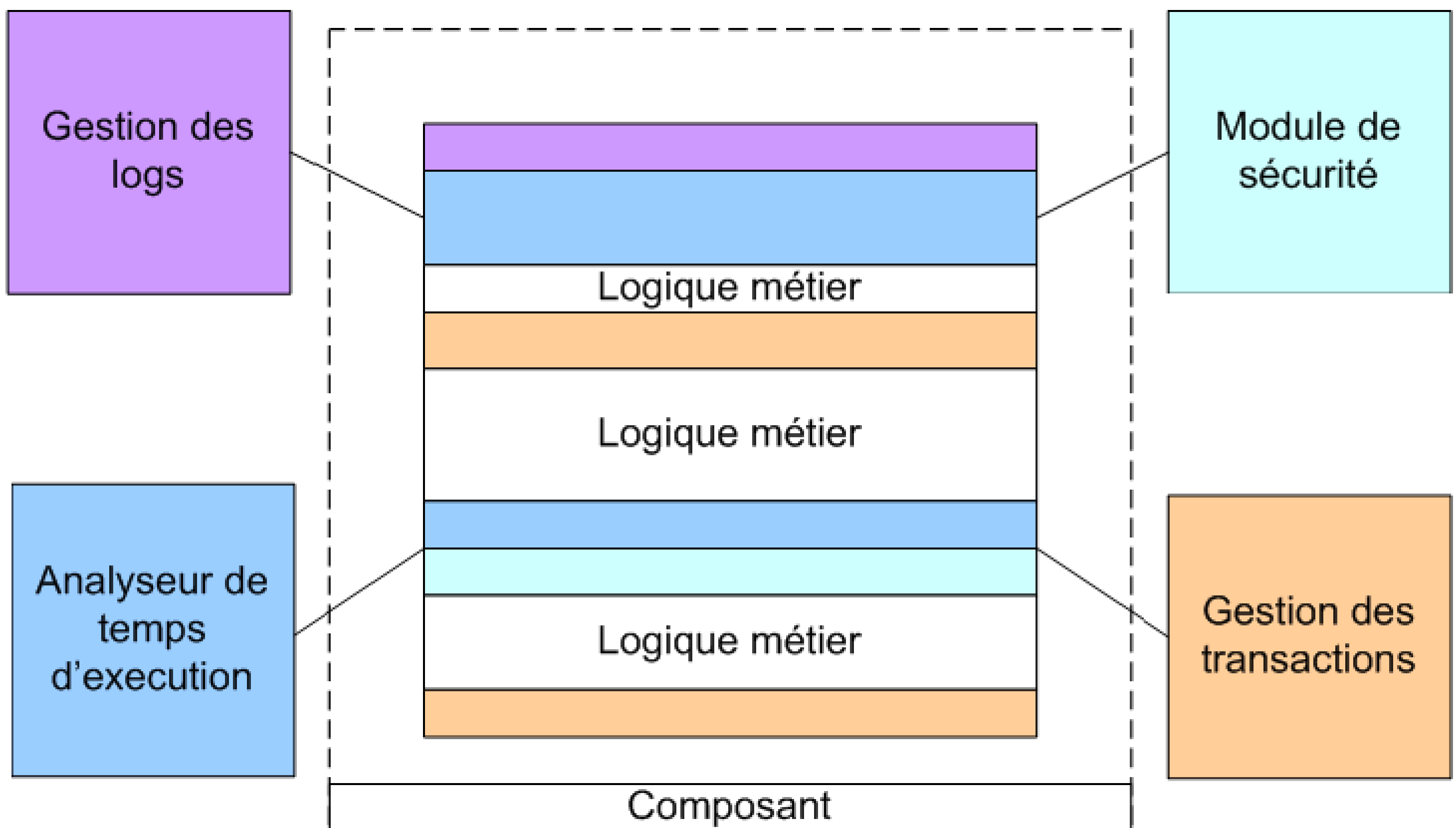
Atelier 3 : Programmation Orientée Aspect (AOP)

Atelier 3 - Programmation Orientée Aspect (AOP)

Principe

La programmation orientée aspect (en anglais **aspect-oriented programming - AOP**) est un paradigme de programmation qui permet de séparer les considérations techniques (**aspect** en anglais) des descriptions métier dans une application. Par exemple, le principe de l'inversion de contrôle (en anglais, **IOC**, Inversion Of Control) peut être implémentée par cette méthode de programmation.

Comme nous pouvons le voir dans la figure suivante, un module ou composant métier est régulièrement pollué par de multiples appels à des composants utilitaires externes.



De fait ces appels rendent le code plus complexe et donc moins lisible. Comme chacun sait, un code plus court et donc plus clair améliore la qualité et la réutilisabilité.

La solution constituerait donc à externaliser tous les traitements non relatifs au logique métier en dehors du composant. Pour ce faire il faut pouvoir définir des traitements de façon déclarative ou programmatique sur les points clés de l'algorithme. Typiquement avant ou après une méthode.

Dans la plupart des cas ce genre de traitements utilitaires se fait en début ou en fin de méthode, comme par exemple **journaliser** les appels ou encore effectuer un commit ou un rollback sur une transaction.

La démarche est alors la suivante:

- **Décomposition en aspect**: Séparer la partie métier de la partie utilitaire.
- **Programmation de la partie métier**: Se concentrer sur la partie variante.
- **Recomposition des aspects**: Définition des aspects

Grâce à l'AOP, le couplage entre les modules gérant des aspects techniques peut être réduit de façon très importante, ce qui présente de nombreux avantages :

- Maintenance accrue
- Meilleure réutilisation
- Gain de productivité
- Amélioration de la qualité du code

Lexique

La **programmation orientée aspect** définie un jargon bien spécifique, simples mais puissants :

- **aspect** : un module définissant des greffons et leurs points d'activation.
- **greffon** (en anglais, **advice**) : un programme qui sera activé à un certain point d'exécution du système, précisé par un point de jonction.
- **tissage** ou **tramage** (en anglais, **weaving**) : insertion statique ou dynamique dans le système logiciel de l'appel aux greffons.
- **point d'action**, de **coupure**, de **greffe** (en anglais, **pointcut**) : endroit du logiciel où est inséré un greffon par le tisseur d'aspect.
- **point de jonction**, d'**exécution** (en anglais, **join point**) : endroit spécifique dans le flot d'exécution du système, où il est valide d'insérer un greffon.
- **considérations entrecroisées**, **préoccupations transversales** (en anglais, cross-cutting concerns) : mélange, au sein d'un même programme, de sous-programmes distincts couvrant des aspects techniques séparés.

Exemple d'utilisation

L'objectif de cet atelier est de tracer (Log) les appels aux méthodes dans une application, sans ajouter du code de Log dans cette application. Grace à Spring, il est possible de tracer les appels aux méthodes objets pendant l'exécution et **sans modifier** le code de l'application.

Tout d'abord, vous devez créer un 'projet Spring' à l'aide du plugin eclipse Spring IDE (voir Lab Spring IDE).

Ajouter les jars suivants à votre application :

- **spring.jar** présent dans \SPRING_HOME\dist
- **commons-logging.jar** présent dans \SPRING_HOME\ lib\jakarta-commons
- **aspectjweaver.jar** présent dans \SPRING_HOME\ lib\aspectj
- **aspectjrt.jar** présent dans \SPRING_HOME\ lib\aspectj
- **cglib-nodep-2.1_3.jar** présent dans \SPRING_HOME\ lib\cglib

Avec « SPRING_HOME » désigne le chemin d'installation du framework spring.

AOP classique

Soit l'application **Calculatrice** qui permet de faire l'addition ou la soustraction de deux entier. Le principe est d'intercepter les entrées/sorties des méthodes "addition(int,int)" et "soustraction(int,int) " et de les logger à l'aide d'une classe externe "**CalculatriceLogger**".

La classe Calculatrice

La classe **Calculatrice** à tracer est la suivante :

```
package org.insat.aop;

public class Calculatrice {

	public int addition(int a, int b) {
		return a + b;
	}

	public int soustraction(int a, int b) {
		return a - b;
	}

}
```

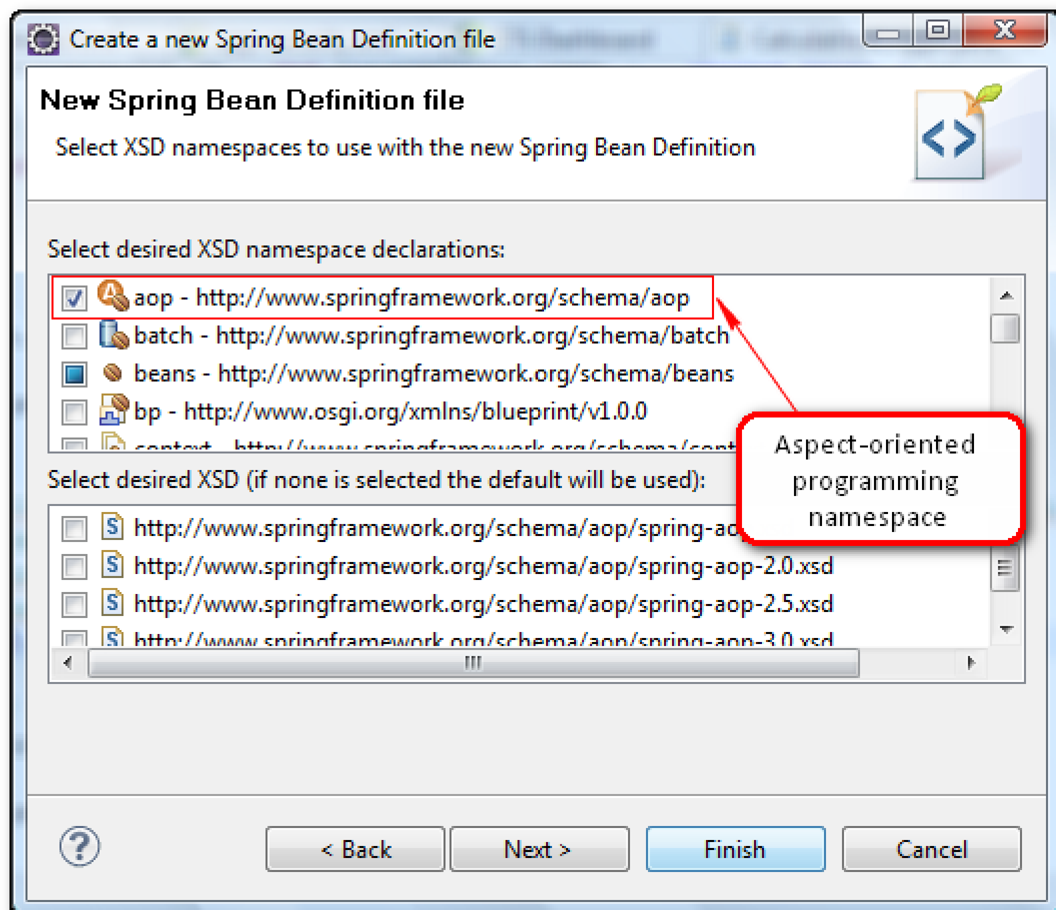
La classe CalculatriceLogger

La classe **CalculatriceLogger** contient les deux méthodes "**logMethodEntry()**" et "**logMethodExit()**" qui vont être appelées par **Spring-AOP** pour tracer les appels des méthodes interceptées. Le code de cette classe est le suivant :

```
package org.insat.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.JoinPoint.StaticPart;

public class CalculatriceLogger {

	// Cette méthode est appelée à chaque fois (et avant) qu'une méthode du
	// package org.insat.aop est interceptée
	public void logMethodEntry(JoinPoint joinPoint) {

		Object[] args = joinPoint.getArgs();

		// Nom de la méthode interceptée
		String name = joinPoint.getSignature().toLongString();
		StringBuffer sb = new StringBuffer(name + " called with: [");

		// Liste des valeurs des arguments reçus par la méthode
		for (int i = 0; i < args.length; i++) {
			Object o = args[i];
			sb.append("'" + o + "'");
			sb.append((i == args.length - 1) ? "" : ", ");
		}
		sb.append("]");

		System.out.println(sb);
	}

	// Cette méthode est appelée à chaque fois (et après) qu'une méthode du
	// package org.insat.aop est interceptée
	// Elle reçoit en argument 'result' qui est le retour de la méthode
	// interceptée
	public void logMethodExit(StaticPart staticPart, Object result) {

		// Nom de la méthode interceptée
		String name = staticPart.getSignature().toLongString();

		System.out.println(name + " returning: [" + result + "]");
	}

}
```

Configuration AOP

La dernière tâche qui reste est de configurer l'AOP à l'aide d'un fichier de configuration Spring. Donc, créer un fichier de définition des beans, le nommer « springContext.xml ». Lors de la création de ce dernier, il faut ajouter le namespace **AOP** comme suit:



Le fichier de définition des beans est :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

      <bean name="calculatrice" class="org.insat.aop.Calculatrice" />

      <!--  Debut de la configuration AOP -->
      <aop:config>
            <aop:pointcut id="servicePointcut"
                  expression="execution(* org.insat.aop.Calculatrice.*(..))" />
            <aop:aspect id="loggingAspect" ref="calculatriceLogger">
                  <aop:before method="logMethodEntry" pointcut-ref="servicePointcut"
/>
                  <aop:after-returning method="logMethodExit"
                        returning="result" pointcut-ref="servicePointcut" />
            </aop:aspect>
      </aop:config>

      <bean id="calculatriceLogger" class="org.insat.aop.CalculatriceLogger" />
      <!--  Fin de la configuration AOP -->

</beans>
```

Explication de la configuration AOP:

<aop:config> ... </aop:config>
Définit le bloc de configuration AOP.

<aop:pointcut id="servicePointcut" expression="execution(*org.insat.aop.Calculatrice.*(..))"/>
Permet de définir des points d'interception sur les objets.
Ici l'expression org.insat.aop.Calculatrice.* signifie que toutes les méthodes des objets qui sont dans la classe **Calculatrice** seront interceptées.

<aop:aspect id="loggingAspect" ref="calculatriceLogger ">
Les appels aux méthodes seront renvoyés vers le bean Spring " calculatriceLogger " (classe CalculatriceLogger)

<aop:before method="logMethodEntry" ...>
Avant l'appel (before) des deux méthodes "logMethodEntry()" et "logMethodExit()", la méthode "CalculatriceLogger.logMethodEntry()" est appelée.

<aop:after-returning method="logMethodExit" returning="result"...>

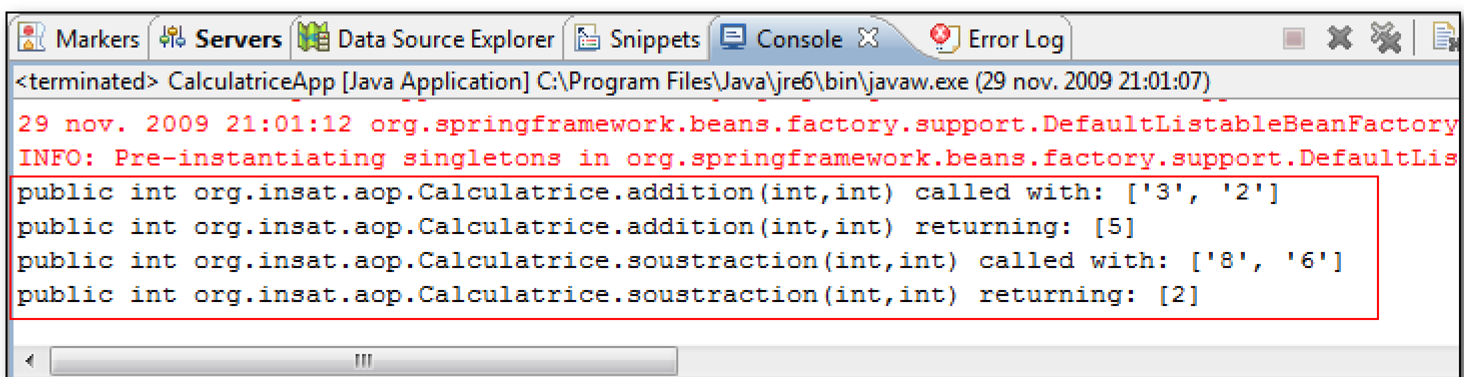
Après l'appel (after) des deux méthodes "logMethodEntry()" et "logMethodExit()", la méthode "CalculatriceLogger.logMethodExit()" est appelée et le résultat des méthodes lui seront passés en argument.

Programme principale

Le programme principal à exécuter est le suivant :

```
package org.insat.main;

import org.insat.aop.Calculatrice;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class CalculatriceApp {

	public static void main(String[] args) {
		ApplicationContext context = new
				ClassPathXmlApplicationContext("springContext.xml");
		Calculatrice calculatrice = (Calculatrice)context.getBean("calculatrice");
		calculatrice.addition(3, 2);
		calculatrice.soustraction(8, 6);
	}

}
```

L'exécution de ce programme résulte :



Atelier 4 :

JPA managé par Spring

Atelier 4 - JPA managé par Spring

Principe

L'utilisation de JPA dans un environnement non managé peut se révéler délicate et problématique.

Ceci est dû essentiellement à la gestion de la session de persistance, qui dans le mode non-managé doit être gérée à la main par le développeur, or la méthode la plus simple qui consiste à ouvrir une session de persistance chaque fois qu'on en a besoin est inefficace.

C'est pour cela qu'il vaut mieux (faut) utiliser un conteneur pour la gestion de la session de persistance (JPA, Hibernate, etc.) comme par exemple le conteneur Spring.

Exemple d'utilisation

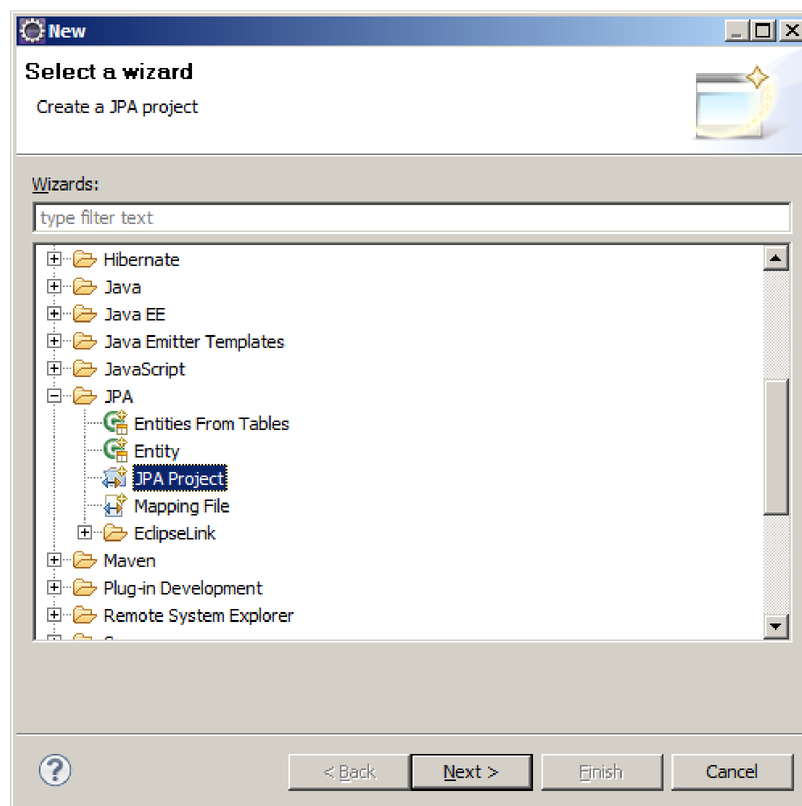
Nous allons présenter rapidement dans ce billet les étapes à suivre pour configurer Spring 2.5 et JPA dans le cadre d'une application java simple (ne nécessitant pas un serveur d'application).

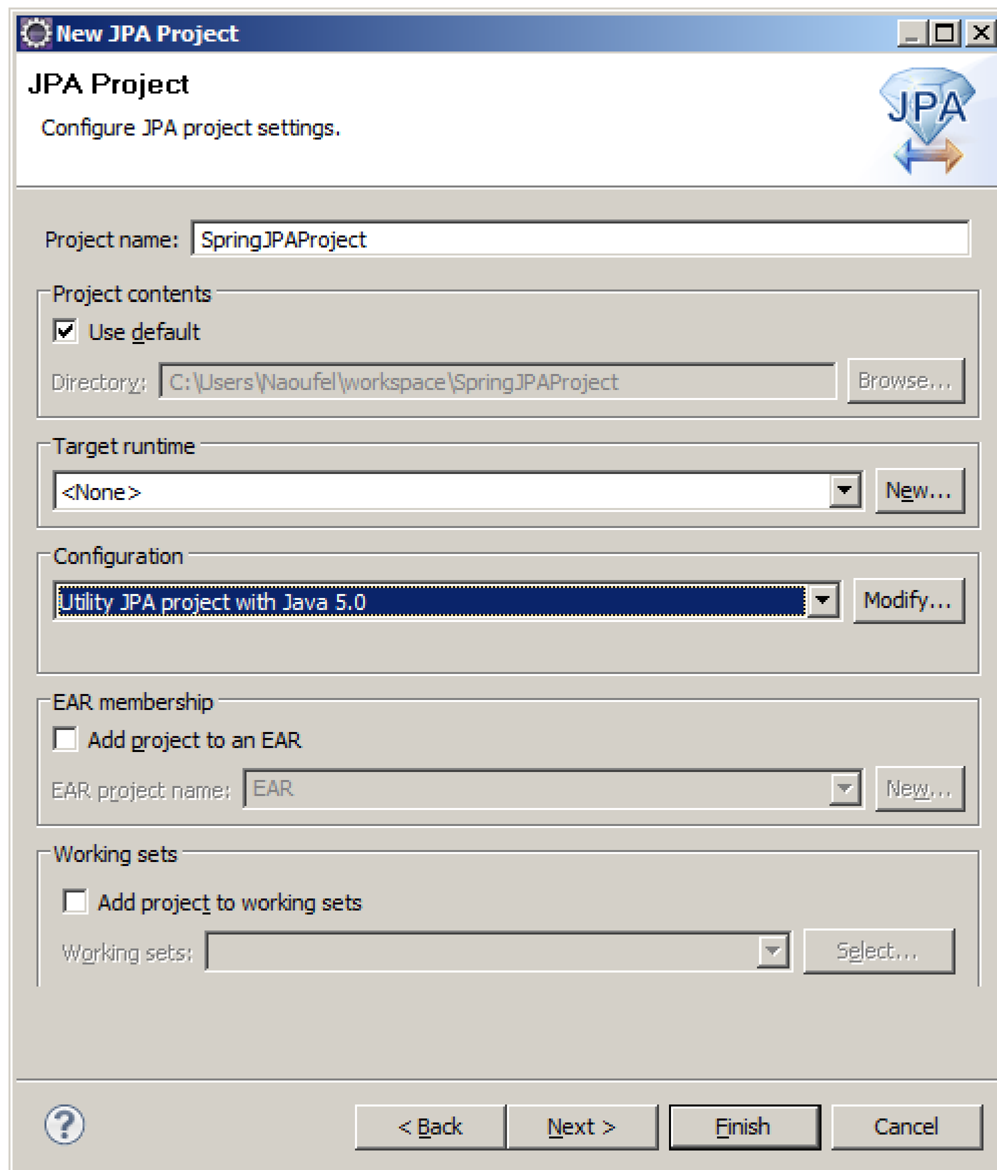
Nous allons utiliser MySQL comme base de données. Il est aussi facile de migrer vers un autre SGBD.

Création du projet java

Pour créer un projet de nature JPA et le configurer, nous allons utiliser le plugin Dali intégré par défaut dans la dernière version d' « Eclipse Galileo » destinée aux développeurs java EE.

Créez un projet de nature JPA comme suit :

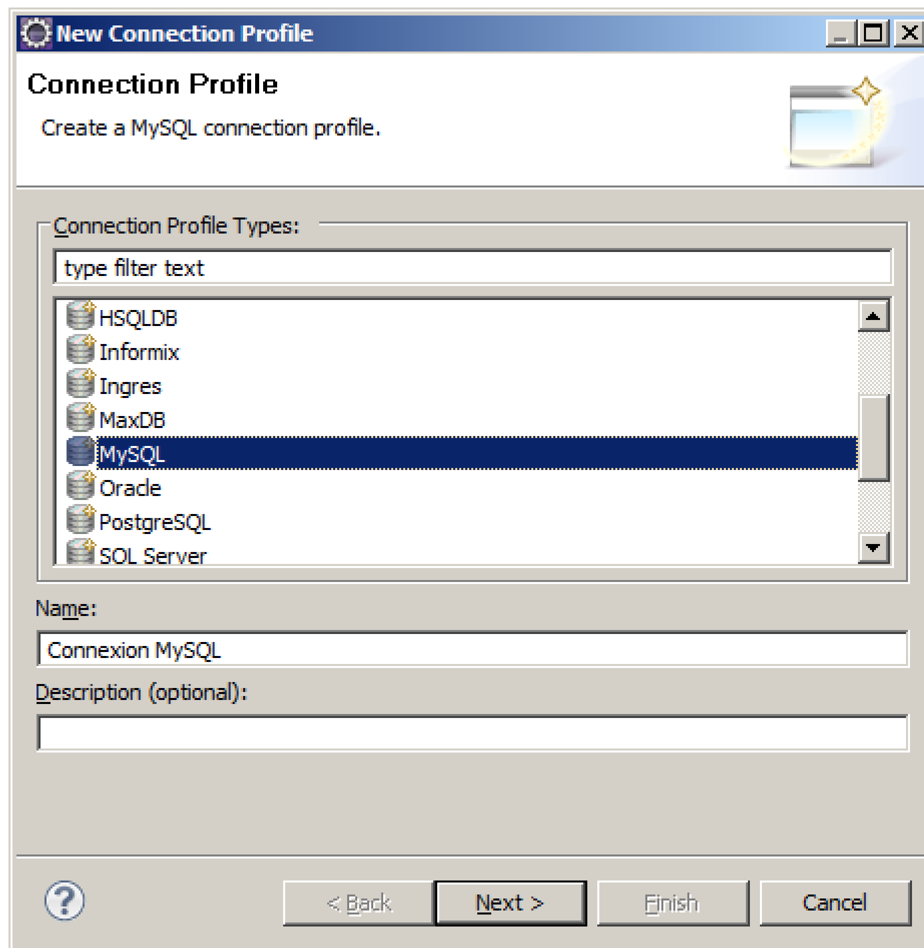




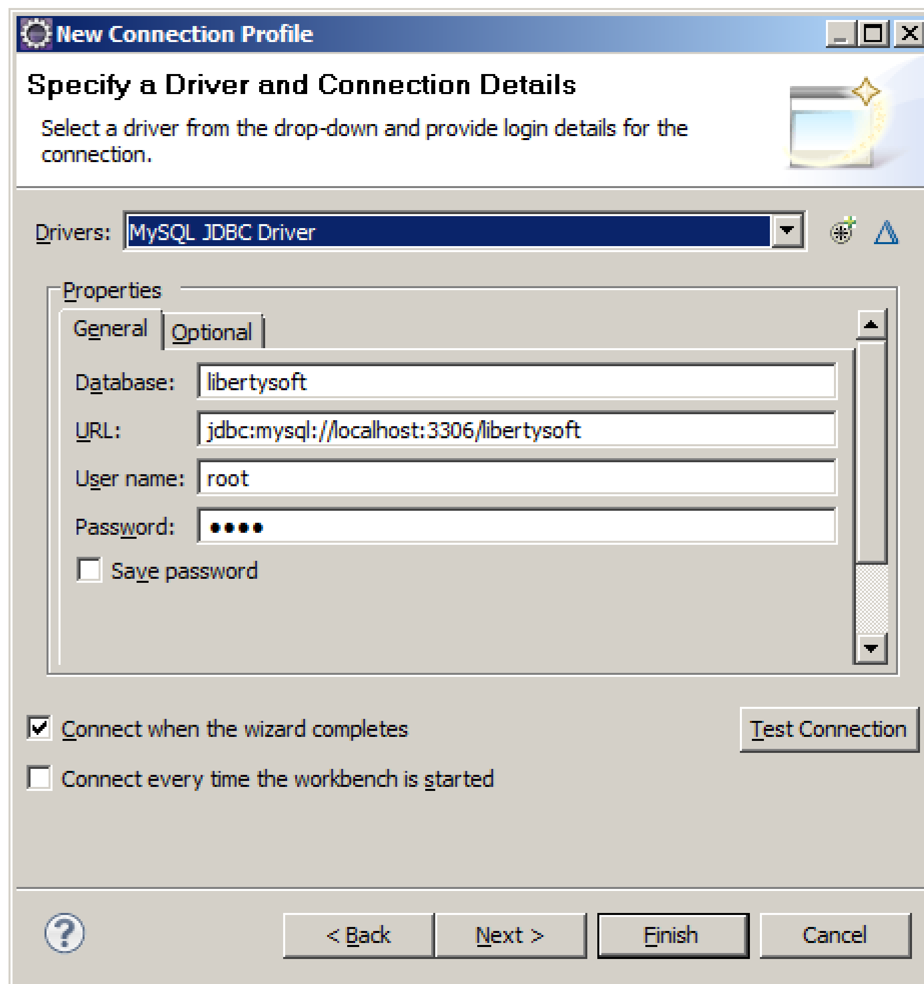
Nous allons maintenant créez la connexion à la base de données de type MySQL :

- cliquez sur « Add Connection … ».

- choisissez le type de connexion MySQL et lui attribuez un nom puis cliquez sur « Next »:



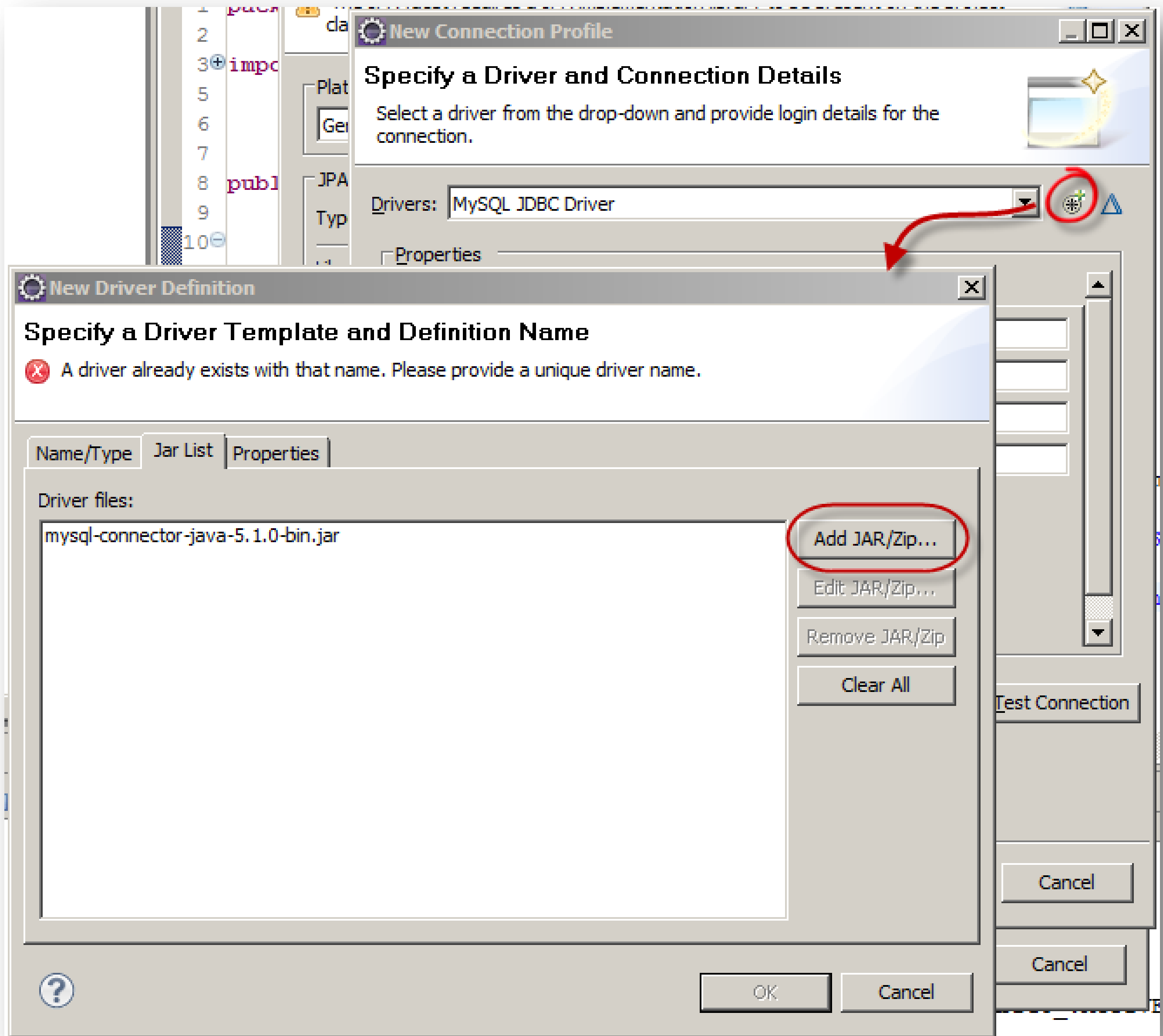
Entrez les paramètres de la base:



Cliquez sur le bouton pour définir le pilote de connexion (*Driver*).

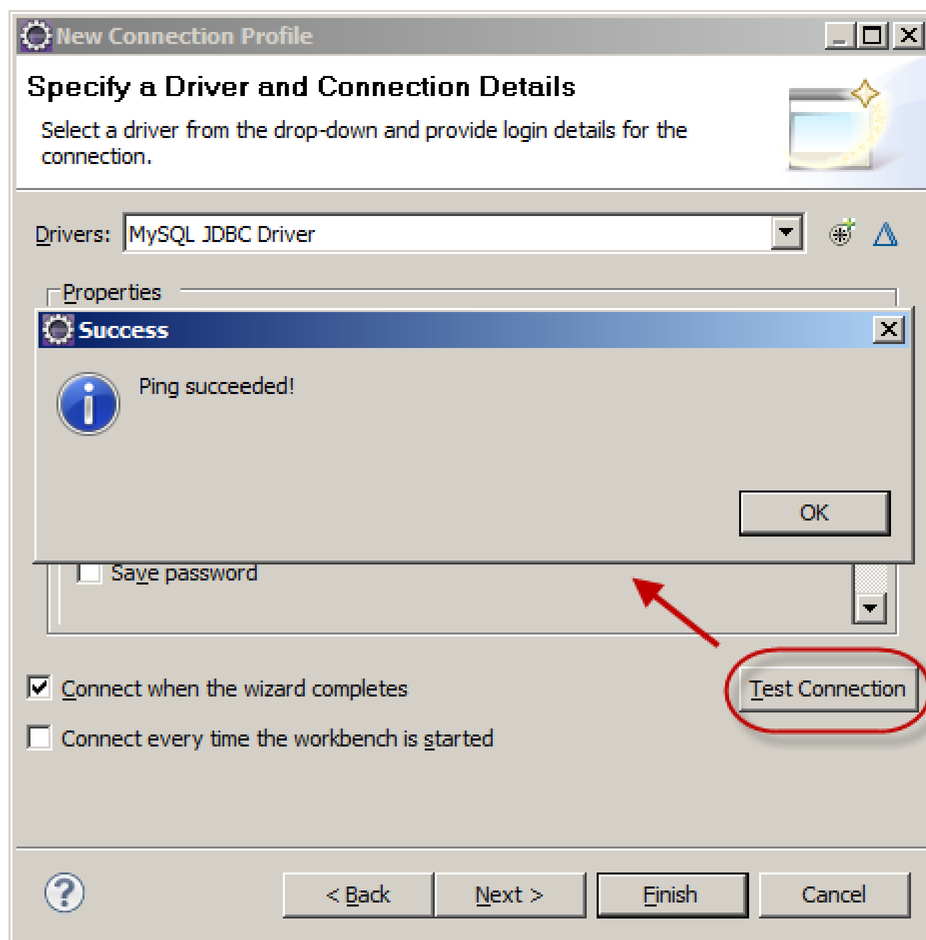
Sous l'onglet « Name/Type » sélectionnez la version 5.1 du Driver puis passez à l'onglet « Jar List ».

Sous l'onglet « Jar List » effacer le jar par défaut et spécifier le chemin du Driver (mysql-connector-java-5.1.9.jar) et cliquez sur « OK ».



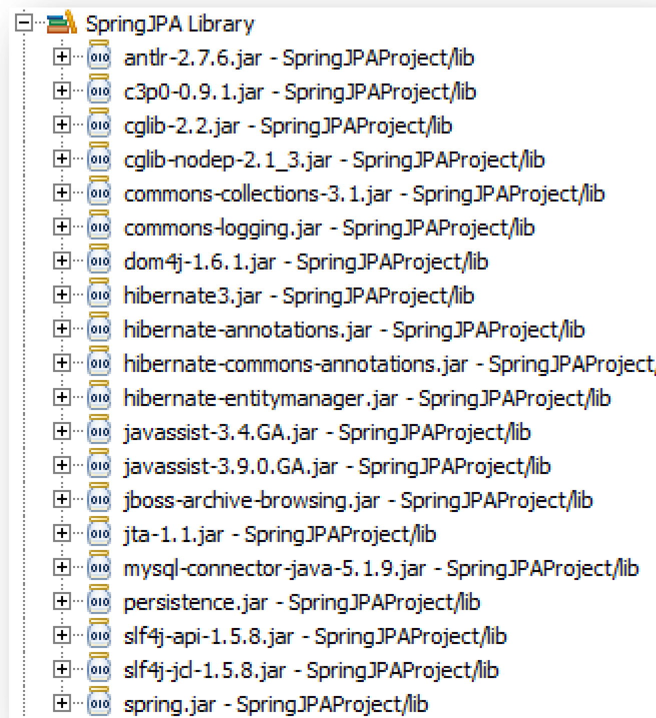
NB : Si le bouton « OK » reste désactivé changez le nom du driver.

Testez enfin la connexion pour être sur que les paramètres entrés sont correctes :



www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

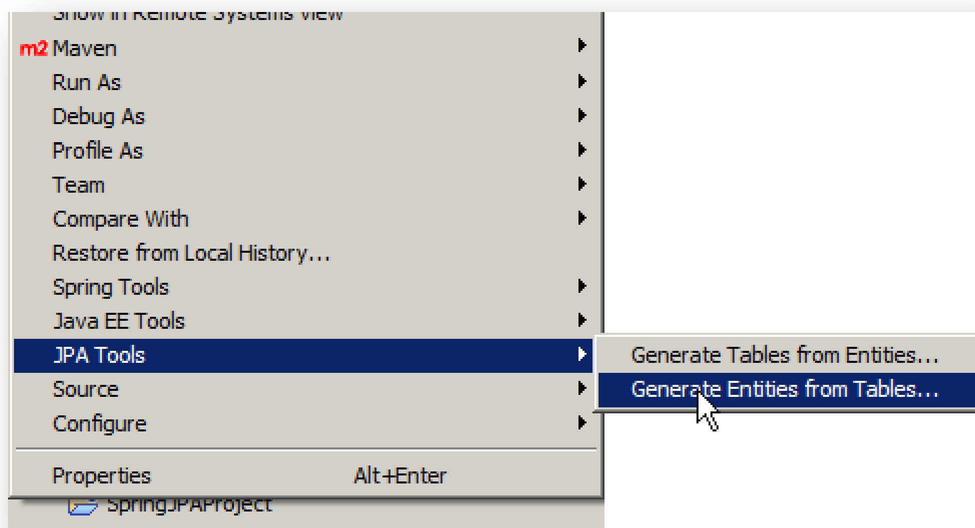
Une fois la procédure est terminée, ajoutez les dépendances suivantes au projet (recourez vous à l'atelier « Spring IDE » section « Ajout des librairies Spring » pour savoir comment ajouter les dépendances au projet) :



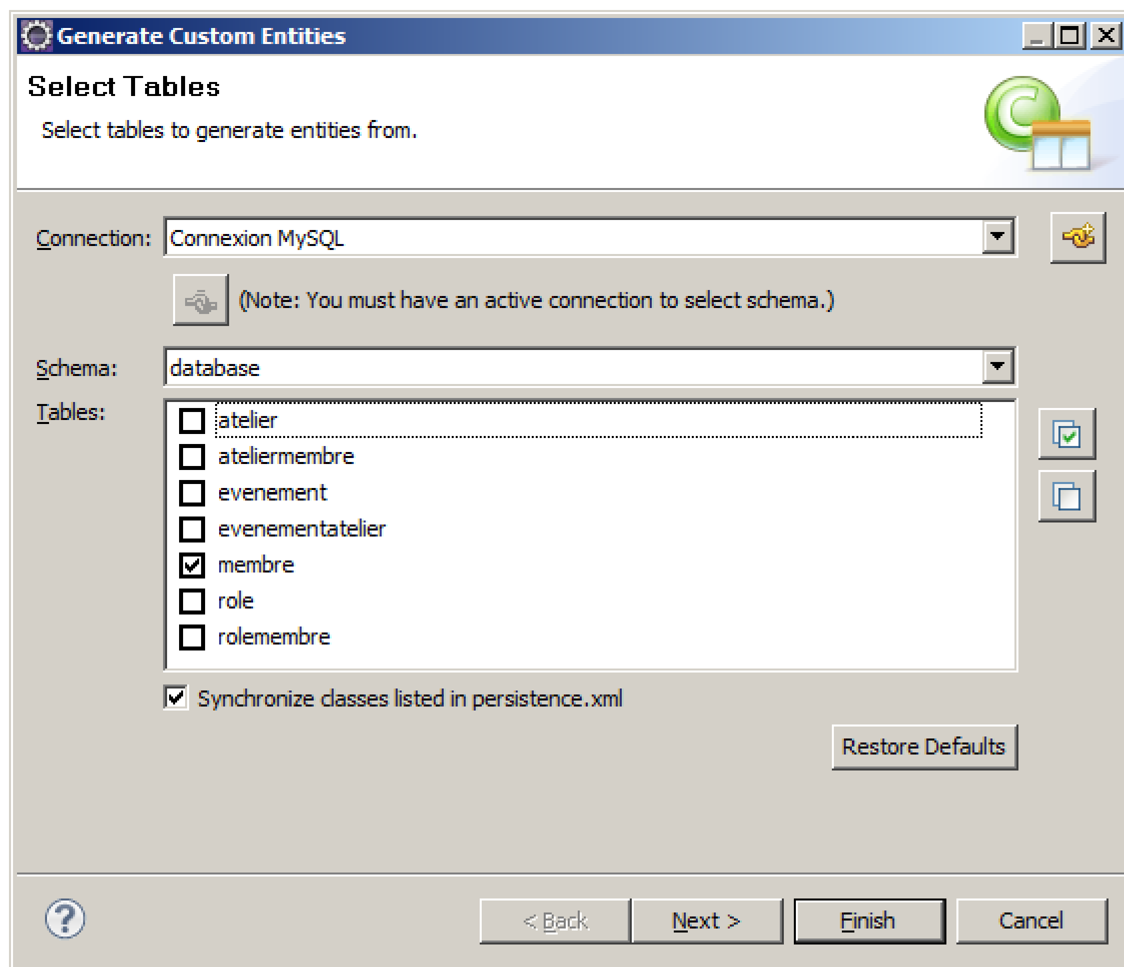
Génération des Entités JPA

Nous allons maintenant créer les entités à partir de la base MySQL.

Cliquez-droit sur le projet puis allez vers *« JPA Tools -> Generate Entities from Tables… »* :



Sélectionnez la connexion déjà créé et choisissez les entités à générer partir des tables:



Sélectionnez le package de destination et cliquez sur « Finish » :

Couche DAO

Nous allons maintenant créer la couche DAO. La classe DAO contient les méthodes CRUD pour accéder à la base de données.

Nous allons définir la classe DAO spécifique à l'entité Membre.

Créez un package « org.insat.dao » et une classe « MembreDAO » et y ajouter le code suivant :

New Java Class

Java Class

Create a new Java class.

Source folder: SpringJPAProject/src Browse...

Package: org.insat.dao Browse...

Enclosing type: Browse...

Name: MembreDAO

Modifiers: public default private protected

abstract final static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

Generate comments

Finish Cancel

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.insat.entities.Membre;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;

@Repository
@Component
public class MemberDAO {

	@PersistenceContext
	private EntityManager entityManagerFactory;

	public Membre findById(String id) {
		return entityManagerFactory.find(Membre.class, id);
	}
}
```

Nous avons spécifiés 2 annotations au sein de cette classe :

- **@Repository**: indique que la classe est un "Repository" (ou Data Access Object - DAO). Avec une certaine configuration dans le fichier « applicationContext.xml » (que nous décrirons plus loin), cette classe sera automatiquement géré par Spring à l'exécution (runtime). Elle sera instanciée et sauvegardée et peut être ensuite injecté dans les classes de service directement.
- **@Component:** indique que le bean est managé par Spring.

Nous avons annoté l' « EntityManager » par **« @PersistenceContext »** pour indiquer à Spring qu'il doit l'injecter automatiquement.

Le reste est du JPA ordinaire.

Couche Métier

Maintenant, nous allons définir la couche métier qui sera appelé par suite.

Pour la couche service, nous allons définir d'abord une interface, qui définit des méthodes communes et ensuite les implémenter.

Dans notre cas, créez un package appelé « org.insat.service » et une interface « MembreService »:

Interface MembreService

```
import org.insat.entities.Membre;

public interface MembreService {

      public Membre findById(String id);

}
```

Créez ensuite la classe d'implémentation « MembreServiceImpl »

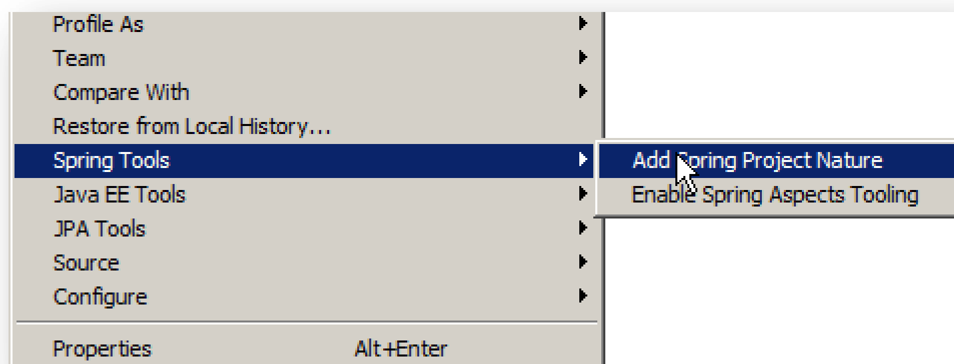
Classe MembreServiceImpl

```
package org.insat.service;

import org.insat.dao.MembreDAO;
import org.insat.entities.Membre;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

@Service
public class MembreServiceImpl implements MembreService {

	@Autowired
	MembreDAO memberDao;

	@Transactional
	public Membre findById(String id) {
		// TODO Auto-generated method stub
		return memberDao.findById(id);
	}

}
```

- Nous avons annoté la classe avec **@Service** pour la marquer en tant que Spring Bean, et plus précisément comme faisant partie de la couche Service.
- Nous avons déclaré le champ « memberDao » annotés avec **@Autowired** pour qu'il soit instancié par l'injection de Spring de manière automatique.
- Nous avons annoté la méthode « finfById » avec **@Transactional**, pour indiquer à Spring qu'il doit l'englober dans une transaction. Ainsi la transaction est soit validée en totalité (commit) soit annulée en totalité en cas d'échec (rollback).

Configuration du projet

Commencez par rendre le projet de nature Spring



Créer ensuite un fichier nommé « jdbc.properties » sous le répertoire « src » pour y stocker les paramètres de la base :

jdbc.properties

```
jdbc.driverClass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/libertysoft?autoReconnect=true
jdbc.username=root
jdbc.password=root
jdbc.database=MYSQL
jdbc.showSql=true
```

Nous allons maintenant indiquer les paramètres de la bd et des composants Bean à travers un fichier de configuration.

Créez le fichier de configuration « applicationContext.xml » sous le répertoire « src » et y insérez le code suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
            http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">


       <context:property-placeholder location="classpath:jdbc.properties" />

       <!-- Connection Pool -->
       <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
             destroy-method="close">
             <property name="driverClass" value="${jdbc.driverClass}" />
             <property name="jdbcUrl" value="${jdbc.url}" />
             <property name="user" value="${jdbc.username}" />
             <property name="password" value="${jdbc.password}" />
       </bean>

       <!-- JPA EntityManagerFactory -->
       <bean id="entityManagerFactory"
             class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
             p:dataSource-ref="dataSource">
             <property name="jpaVendorAdapter">
                   <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
                         <property name="database" value="${jdbc.database}" />
                         <property name="showSql" value="${jdbc.showSql}" />
                   </bean>
             </property>
       </bean>

       <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager"
             p:entityManagerFactory-ref="entityManagerFactory" />

       <context:annotation-config />

       <tx:annotation-driven transaction-manager="transactionManager" />

       <context:component-scan base-package="org.insat.dao" />

</beans>
```

Explication

- Spécifier la DataSource qui est utilisé par Hibernate :

```
<bean id="dataSource" destroy-method="close">
```

- Indiquer à Spring les paramètres de l' « entityManagerFactory » à initialiser :

```
By <bean id="entityManagerFactory" p:dataSource-ref="dataSource">
```

- Demander à Spring de scanner le package en cours et de déterminer les composants à instancier (dans notre cas ce sont les classes « MemberDAO » annotée par « @Repository » et « MemberServiceImpl » annotée par « @Service » qui seront instanciées) :

```
<context:component-scan base-package="com.test.dao"/>
```

- Activer l'AutoWiring : Rechercher @Autowired et injecter l'instance du bean correspondant.

```
<context:annotation-config/>
```

Exécution du projet

Exécuter le code suivant pour tester l'extraction des données à partir de la base :

Classe org.insat.test.TestMembre

```
package org.insat.test;

import org.insat.service.MembreService;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestMembre {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

		MembreService membreService;

		ClassPathXmlApplicationContext appContext = new
ClassPathXmlApplicationContext(new String[] {"applicationContext.xml"});

		membreService = (MembreService) appContext.getBean("membreServiceImpl");

		System.out.println("Nom : " +
membreService.findById("gl@insat.tn").getNommembre());

	}
}
```

```
INFO: building session factory
13 déc. 2009 12:01:01 org.slf4j.impl.JCLLoggerAdapter info
INFO: Not binding factory to JNDI, no JNDI name configured
Hibernate: select membre0_.MAILMEMBRE as MAILMEMBRE0_0_, membr
Nom : GL INSAT
```

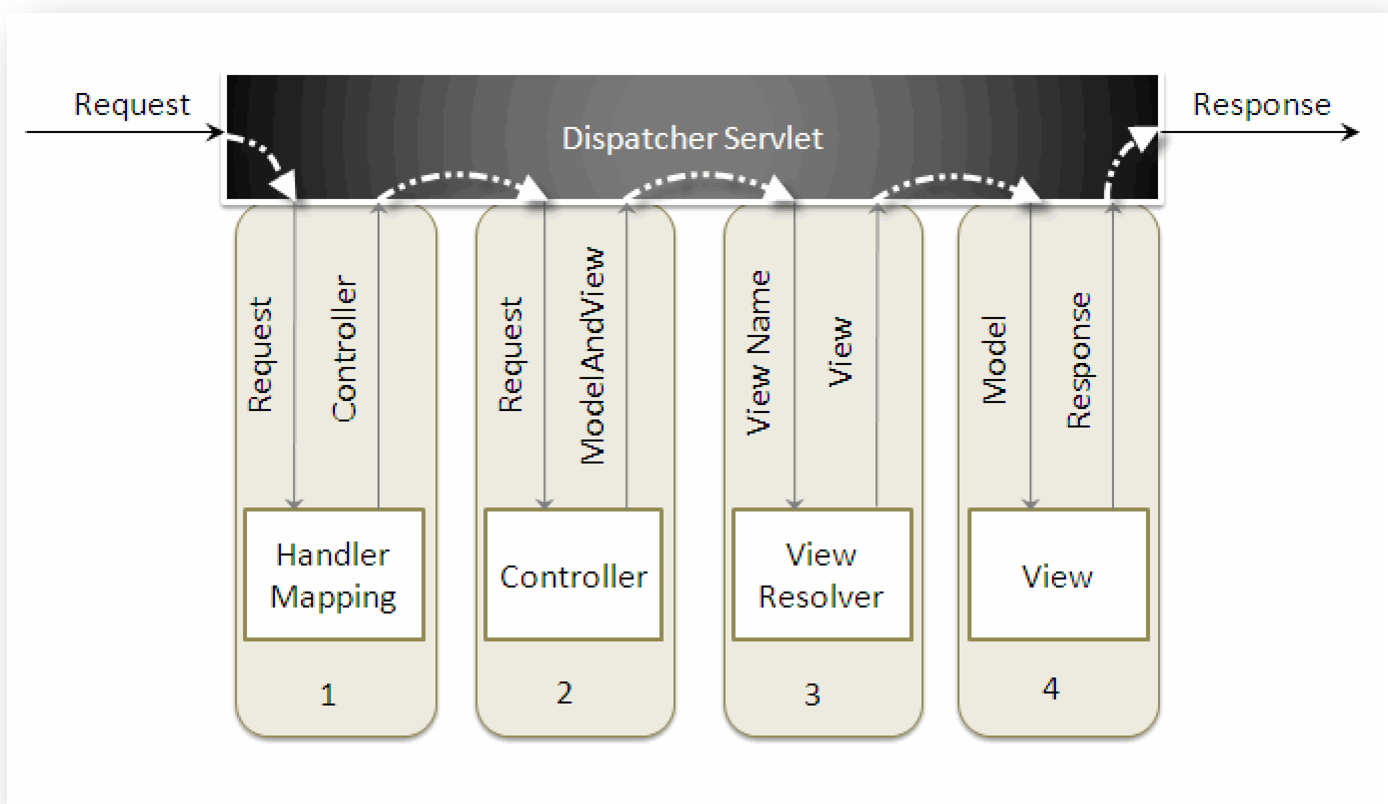
Atelier 5:
Spring MVC

Atelier 5 - Spring MVC

Principe

Spring MVC aide à développer des applications web flexible et faiblement couplés. Le modèle de conception Modèle-Vue-Contrôleur permet de séparer la logique métier, la logique de présentation et la logique de navigation.
Les modèles sont responsables pour encapsuler les données d'application. Les Vues ont comme rôle de rendre réponse à l'utilisateur à l'aide de l'objet modèle. Les contrôleurs sont chargés de recevoir la demande de l'utilisateur et d'appeler des services.

La figure ci-dessous montre le flux de la demande dans le Framework MVC Spring :



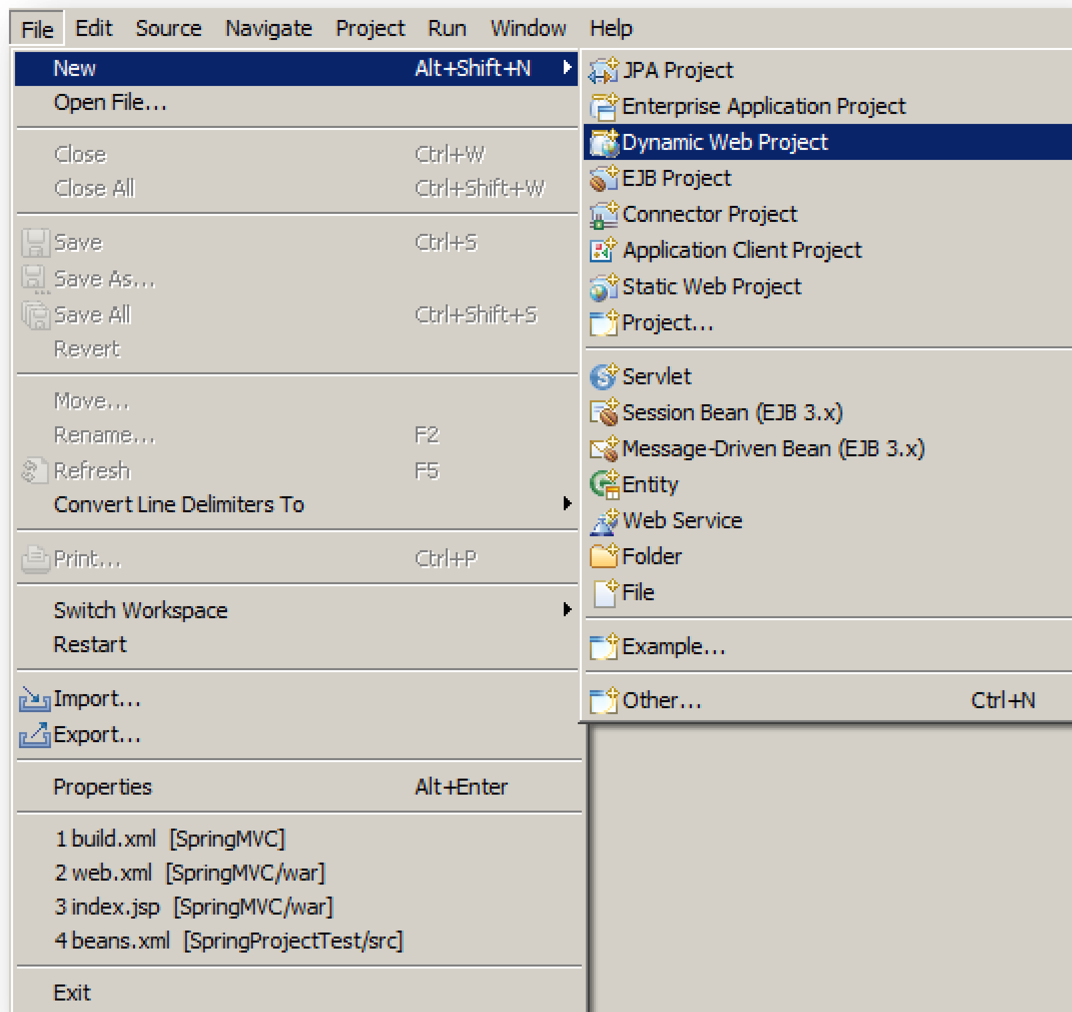
Quand une requête est envoyée, la séquence suivante d'événements se produit.

- Le « DispatcherServlet » reçoit d'abord la demande.
- Le « DispatcherServlet » consulte le « HandlerMapping » et invoque le contrôleur associé à la demande.
- Le contrôleur fait appel aux méthodes appropriés de la couche service et retourne un objet « ModeAndView » au « DispatcherServlet ». L'objet « ModeAndView » contient les données du modèle et le nom de la vue.
- Le « DispatcherServlet » envoie le nom de la vue au « ViewResolver » pour trouver la vue actuelle à invoquer.
- Le « DispatcherServlet » passe l'objet modèle à la vue.
- Enfin, la vue, à l'aide du modèle de données, rend le résultat à l'utilisateur.

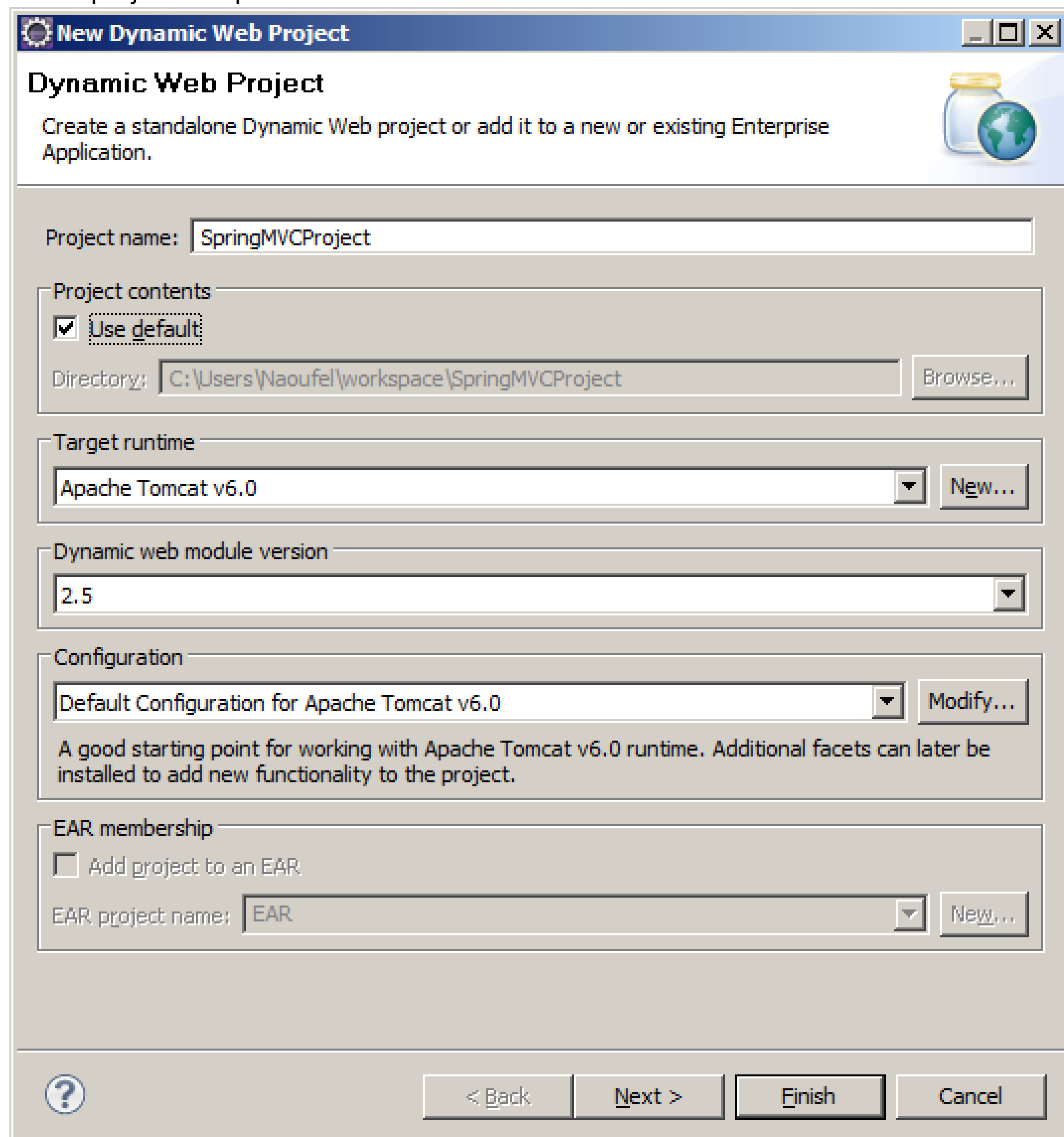
Exemple d'utilisation

L'objet de ce lab est d'illustrer progressivement par un exemple simple mais pratique la création d'une application web avec Spring MVC.

Allez vers « New -> Dynamic Web Project » pour créer un nouveau projet web dynamique :

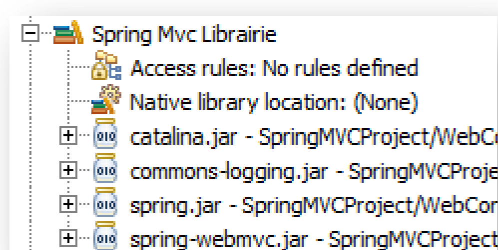


Entrez le nom du projet et cliquez sur Finish :



Cliquez-droit sur le dossier du projet, et sélectionnez « Spring Tools -> Add Spring Project Nature», afin d'ajouter les capacités de Spring au projet Web :

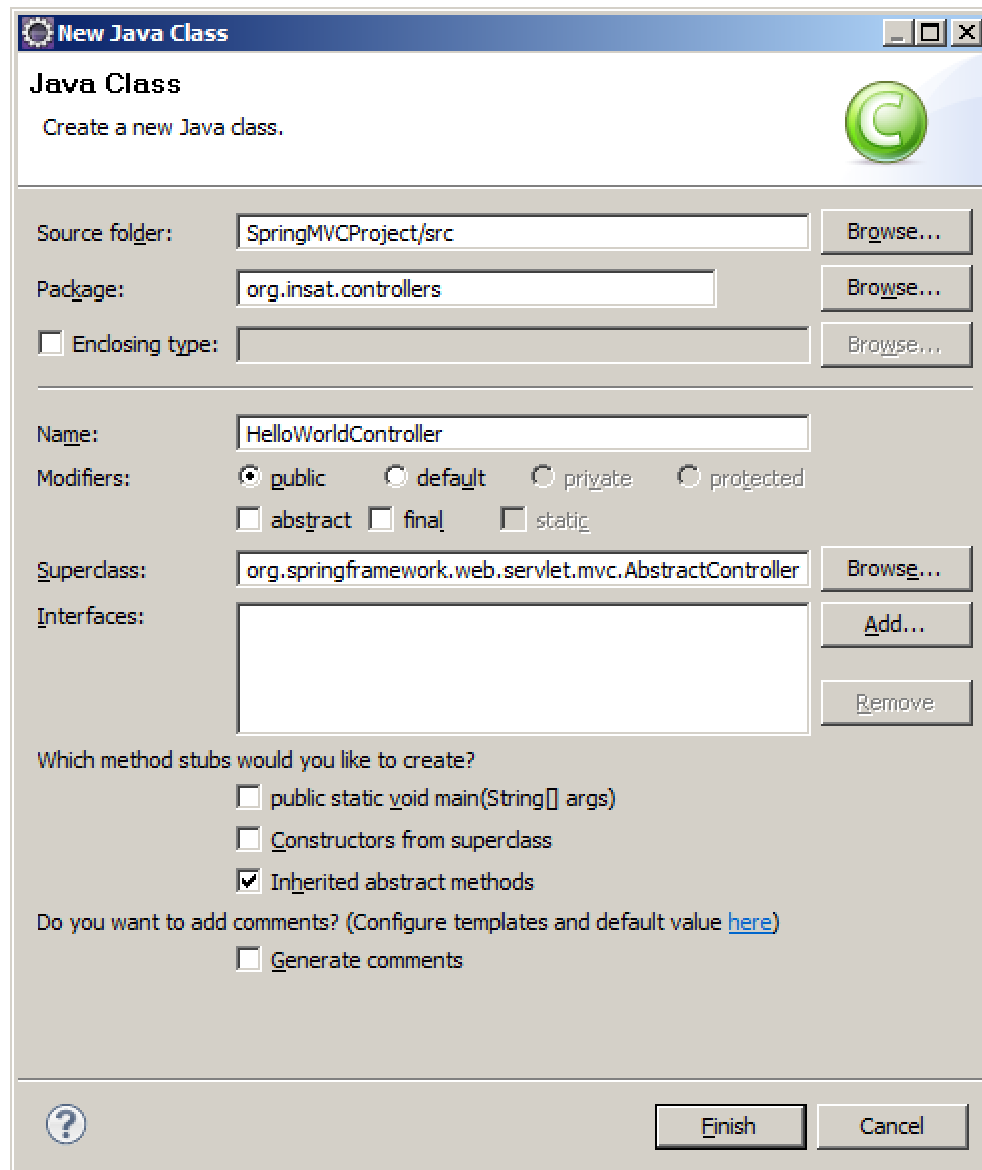
Ajoutez les jars suivants au projet :



spring-webmvc.jar est disponible sous ~\spring-framework-2.5.6.SEC01\dist\modules\
spring.jar est disponible sous ~\spring-framework-2.5.6.SEC01\dist \
commons-logging.jar est disponible sous ~\spring-framework-2.5.6.SEC01\lib\jakarta-commons\

Ajout d'un contrôleur

Créez un nouveau contrôleur sous le package « org.insat.controllers ». Une classe contrôleur de Spring doit hériter de « **org.springframework.web.servlet.mvc.AbstractController** ».



Insérez le code suivant à la classe contrôleur ainsi créé :

```
package org.insat.controllers;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

public class HelloWorldController extends AbstractController {

	private String message;

	@Override
	protected ModelAndView handleRequestInternal(HttpServletRequest arg0,
			HttpServletResponse arg1) throws Exception {
		return new ModelAndView("welcomePage", "welcomeMessage", message);
	}

	public void setMessage(String message){
		this.message = message;
	}

}
```

La classe « HelloWorldController » possède l'attribut « message » qui sera défini par le « setter injection ». Cette classe doit implémenter la méthode « handleRequestInternal() » pour traiter la requête. Après traitement de la requête, elle retourne un objet « ModelAndView » au « DispatcherServlet ».

Configuration du « DispatcherServlet »

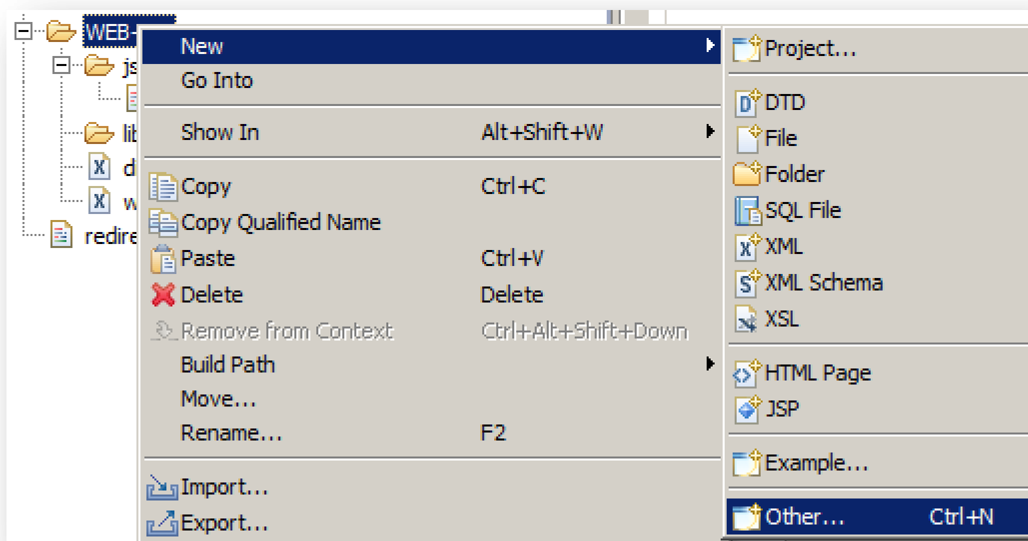
Le « DispatcherSevlet », comme son nom l'indique, est une servlet unique qui gère le processus de traitement des requêtes (request-handling).
Quand une requête est envoyée au « DispatcherServlet », ce dernier délègue le travail en invoquant les contrôleurs concernés pour traiter la demande.
Comme tout autre servlet, le « DispatcherServlet » a besoin d'être configuré dans le descripteur de déploiement Web. Pour ceci ajouter le code suivant au web.xml :

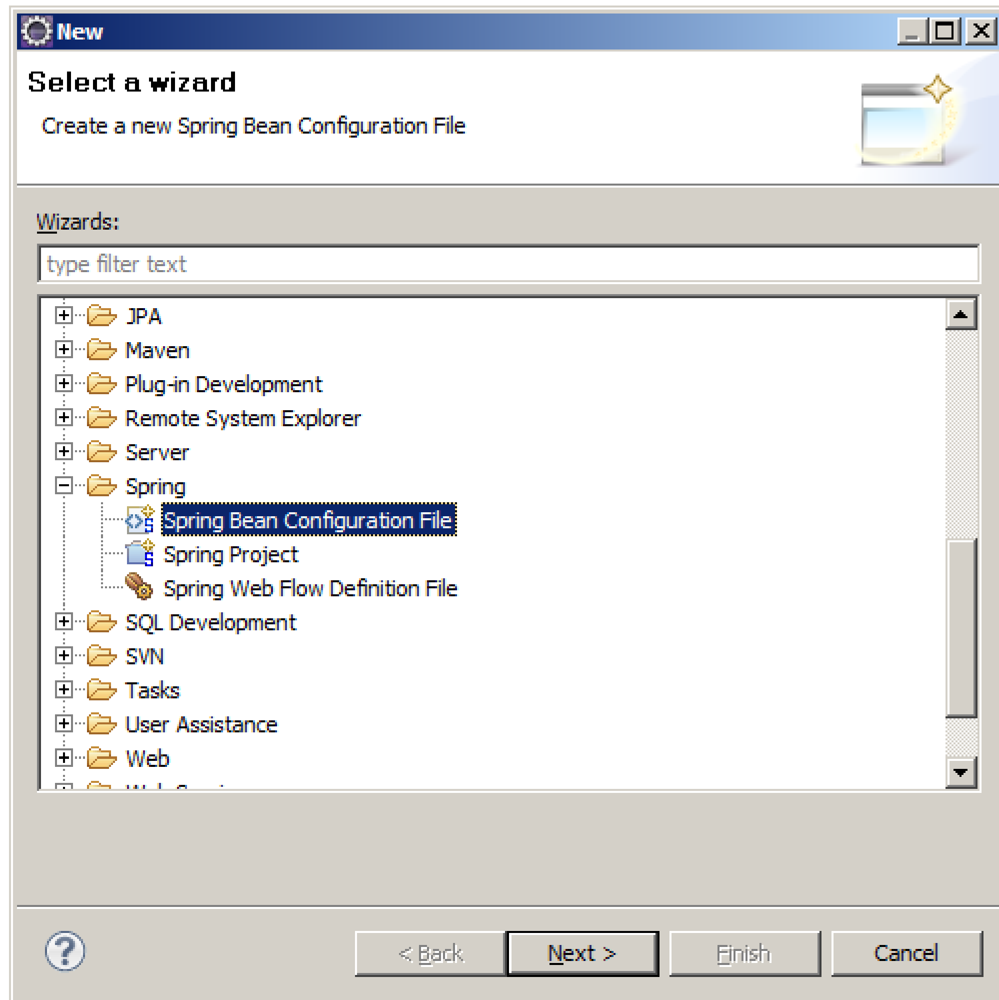
```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>

<welcome-file-list>
  <welcome-file>redirect.jsp</welcome-file>
</welcome-file-list>
```

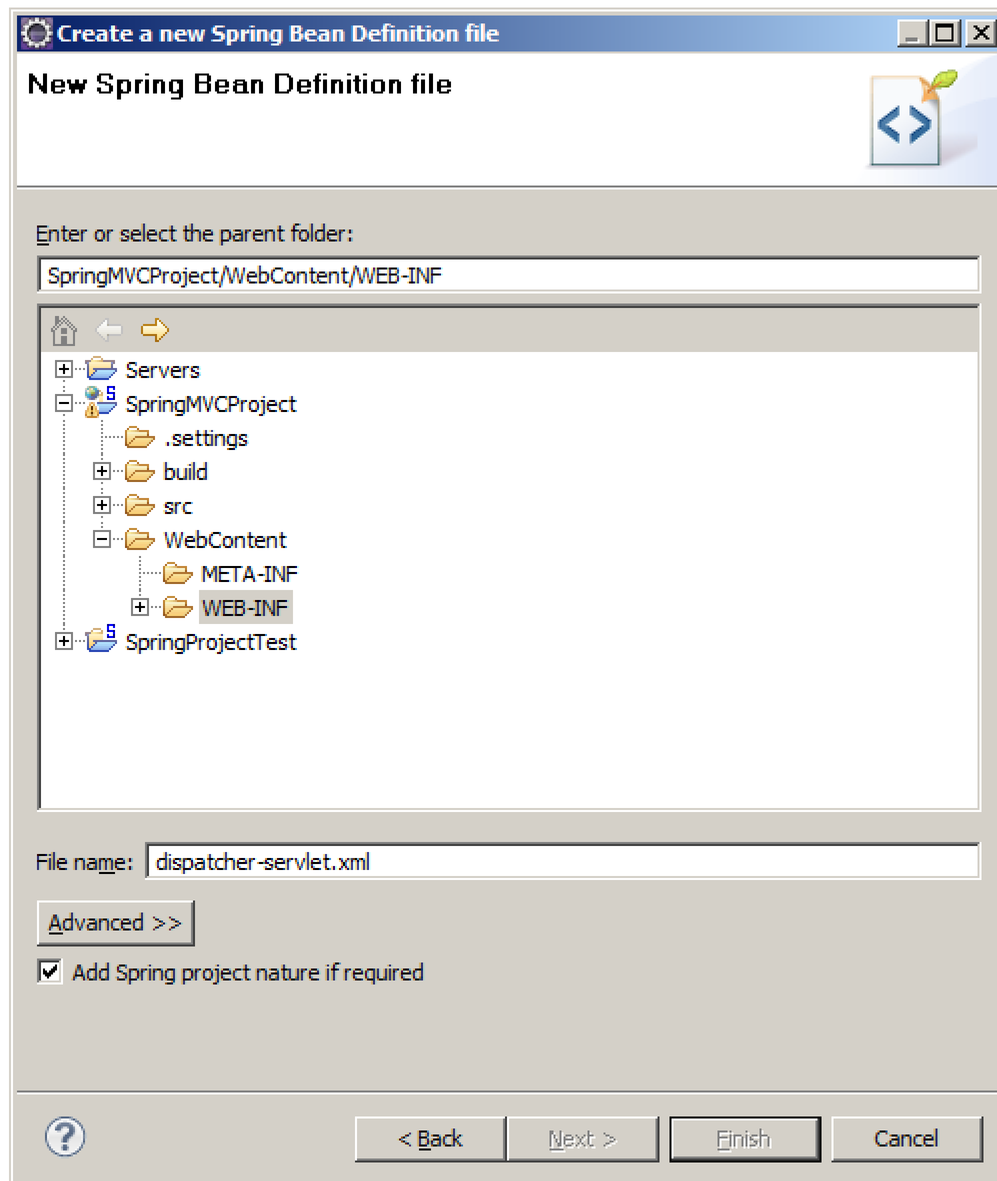
Dans notre cas, le nom du servlet est « dispatcher ». Par défaut, le « DispatcherServlet » cherchera le fichier « dispatcher-servlet.xml » pour charger la configuration de Spring MVC. Ce nom est formée par la concaténation du nom de servlet (« dispatcher ») avec « -servlet.xml ».

Fichier de configuration

Pour créer le fichier de configuration « dispatcher-servlet.xml » cliquez-droit sur le dossier WEB-INF et sélectionnez « New -> Other » et suivez les instructions décrites par les figures suivantes :




New
Select a wizard
Create a new Spring Bean Configuration File
Wizards:
type filter text
JPA
Maven
Plug-in Development
Remote System Explorer
Server
Spring
Spring Bean Configuration File
Spring Project
Spring Web Flow Definition File
SQL Development
SVN
Tasks
User Assistance
Web
< Back
Next >
Finish
Cancel



www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Une fois que le fichier de configuration est créé, nous devons configurer le contrôleur et la classe « ViewResolver ». Ajoutez alors le code suivant au fichier de configuration :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
            <property name="prefix">
                  <value>/WEB-INF/jsp/</value>
            </property>
            <property name="suffix">
                  <value>.jsp</value>
            </property>
      </bean>

      <bean name="/welcome.htm" class="org.insat.controllers.HelloWorldController">
            <property name="message" value="Hello GL INSAT !!"></property>
      </bean>

</beans>
```

Explication du code

- Le contrôleur est configuré par l'extrait XML suivant:

```
<bean name="/welcome.htm" class="org.insat.controllers.HelloWorldController">
      <property name="message" value="Hello GL INSAT !!"></property>
</bean>
```

L'attribut « *name* » indique l'URL de redirection de la requête. Par défaut, le DispatcherServlet utilise le **BeanNameUrlHandlerMapping** pour rediriger les requêtes entrantes. Ainsi, nous n'avons pas besoin d'une configuration supplémentaire.

L'attribut message du contrôleur « HelloWorldController » est injecté via le « *Setter Injection* ».

- Le « ViewResolver » est configuré par le code suivant :

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
            <property name="prefix">
                  <value>/WEB-INF/jsp/</value>
            </property>
            <property name="suffix">
                  <value>.jsp</value>
            </property>
      </bean>
```

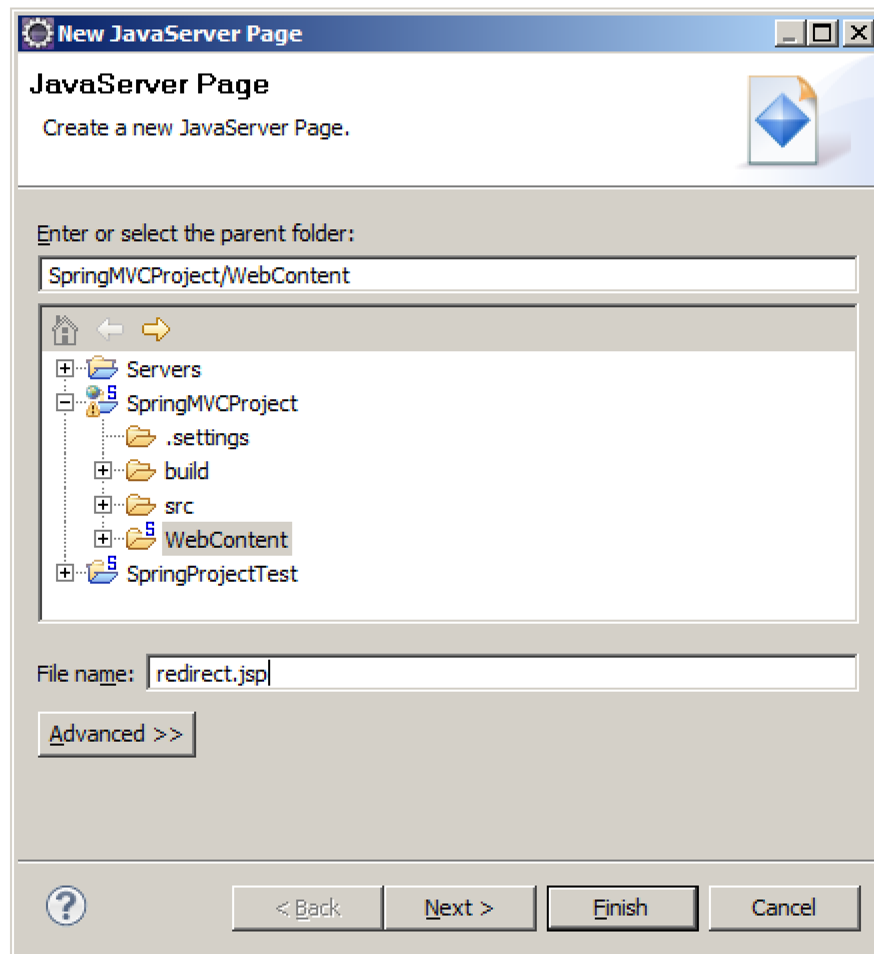
« InternalResourceViewResolver » est utilisé pour résoudre le nom de la vue.

L'emplacement de la vue est donné par la concaténation **valeur du préfixe + nom de la vue + valeur suffixe**.

Dans notre cas, l'emplacement actuel de la vue est /WEB-INF/jsp/welcomePage.jsp

Ajout de la requête (page redirect.jsp)

Insérez une page jsp sous le répertoire « WebContent ». Cette page sera utilisée pour envoyer la requête :



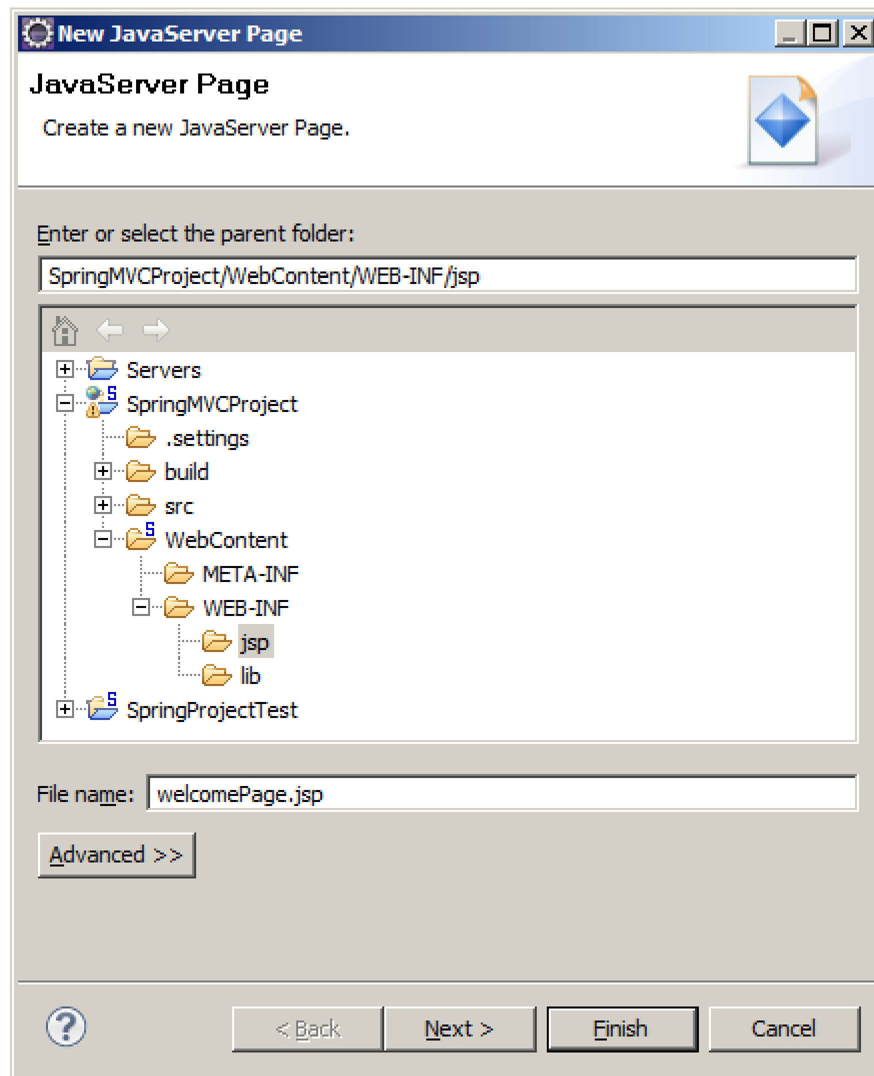
Ajoutez le code suivant à la page redirect.jsp :

```
<%@ page language="java" contentType="text/html;" pageEncoding="UTF-8"%>
<% response.sendRedirect("welcome.htm"); %>
```

La page « redirect.jsp » sera invoquée à l'exécution de l'application Web. Elle va rediriger la requête au « DispatcherServlet », qui à son tour consulte le « BeanNameUrlHandlerMapping » et invoque le « HelloWorldController ». La méthode « handleRequestInternal() » de la classe « HelloWorldController » sera alors invoquée. « handleRequestInternal() » retourne la propriété « message » sous le nom « welcomeMessage » et le nom de la vue « welcomePage » au « DispatcherServlet ». Le « ViewResolver » sera alors invoqué pour déterminer la vue actuelle.

Ajout de la vue

Ajoutez maintenant la page de la vue :

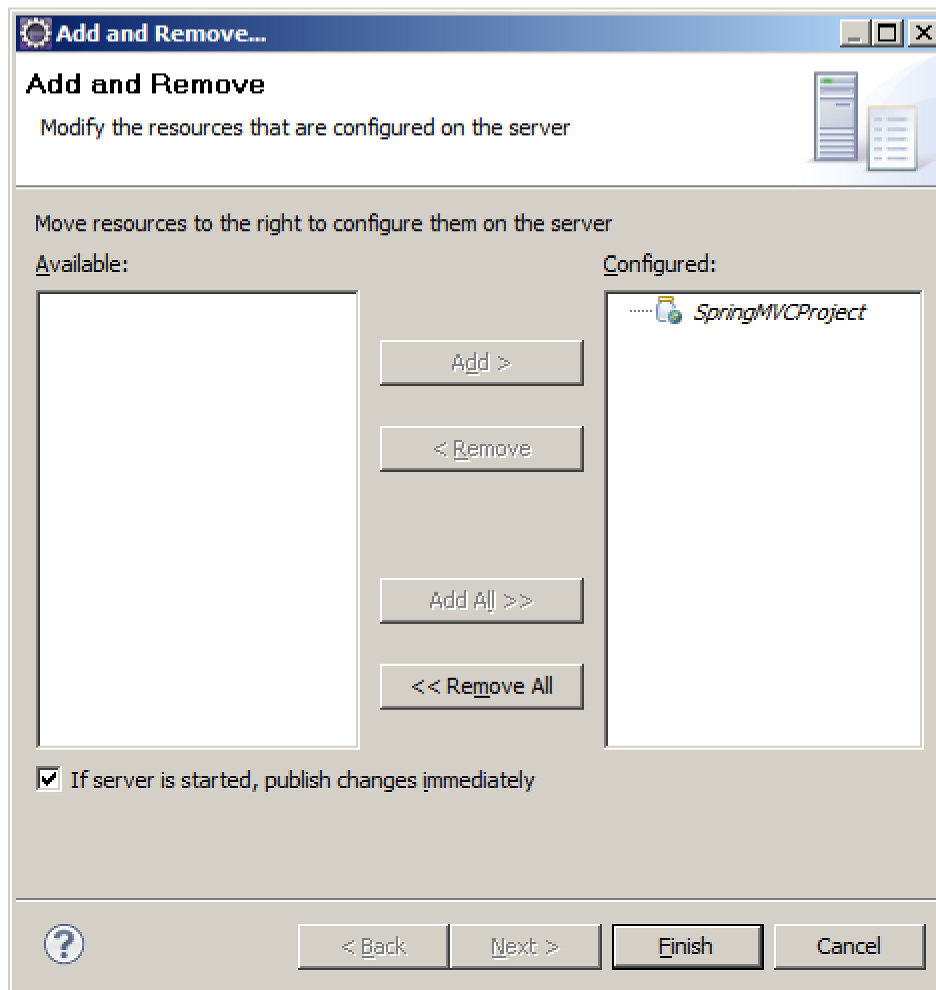


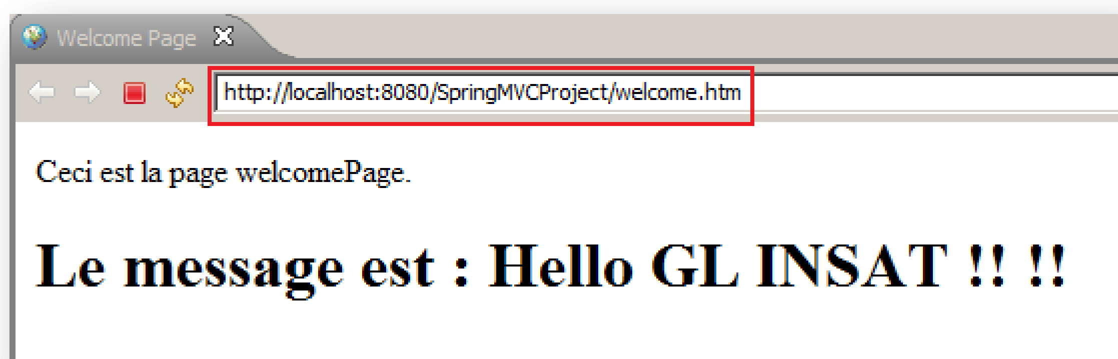

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Welcome Page</title>
</head>
<body>
Ceci est la page welcomePage.
<h1>Le message est : ${welcomeMessage} !!</h1>
</body>
</html>
```

Le paramètre « welcomeMessage » envoyé à partir du contrôleur. Sa valeur est celle de l'attribut message que nous l'avons injecté à travers le fichier de configuration.

Exécution de l'application

Exécuter l'application sous un serveur web (dans notre cas c'est Tomcat 6.0) et accédez à la page redirect.jsp:


Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
SpringMVCProject
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Finish
Cancel


Welcome Page
http://localhost:8080/SpringMVCProject/welcome.htm
Ceci est la page welcomePage.
Le message est : Hello GL INSAT !! !!

Atelier 6 :

Intégration Spring dans JSF

Atelier 6 - Intégration Spring dans JSF

Principe

Spring peut facilement être intégré dans la plupart des Framework web (basé sur java). Tout ce que vous devez le faire est de déclarer un **ContextLoaderListener** dans le fichier **web.xml** et utiliser un **contextConfigLocation <context-param>** pour fixer les fichiers de contexte à charger.

Dans cette approche d'intégration **JSF-Spring**, l'inversion de contrôle est appliqué à la fois par **Spring** et par **JSF**. Le point saillant est que les beans de **Spring** ne sont pas injectés par **Spring**, mais par **JSF**. Donc, **Spring** joue juste le rôle d'un conteneur.

Il est vrai que le support de JSF pour l'injection par setter « setter injection » n'est pas tout à fait différent de celui du Spring. Mais en plus de l'injection de dépendance, Spring peut apporter beaucoup de valeur ajoutée pour JSF, telles que les transactions déclaratives, la sécurité, l'accès distant, etc..., et qui pourraient s'avérer utiles dans une application JSF.

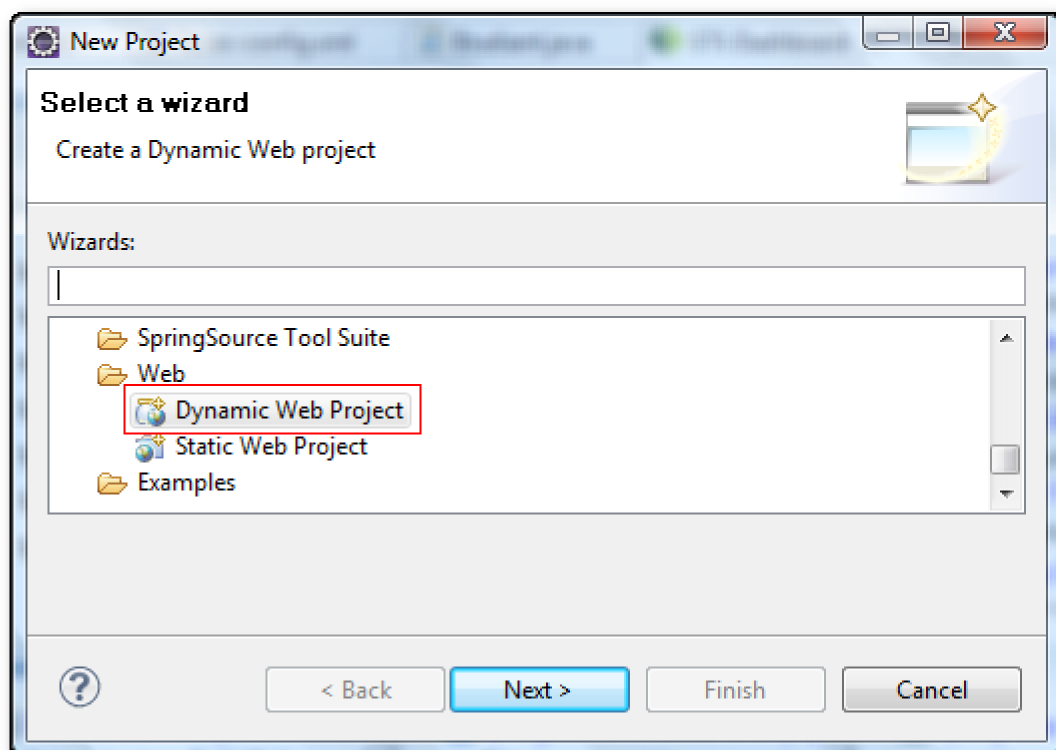
Exemple d'utilisation

La meilleure façon de comprendre l'approche d'intégration **JSF-Spring** est de le voir en exemple.

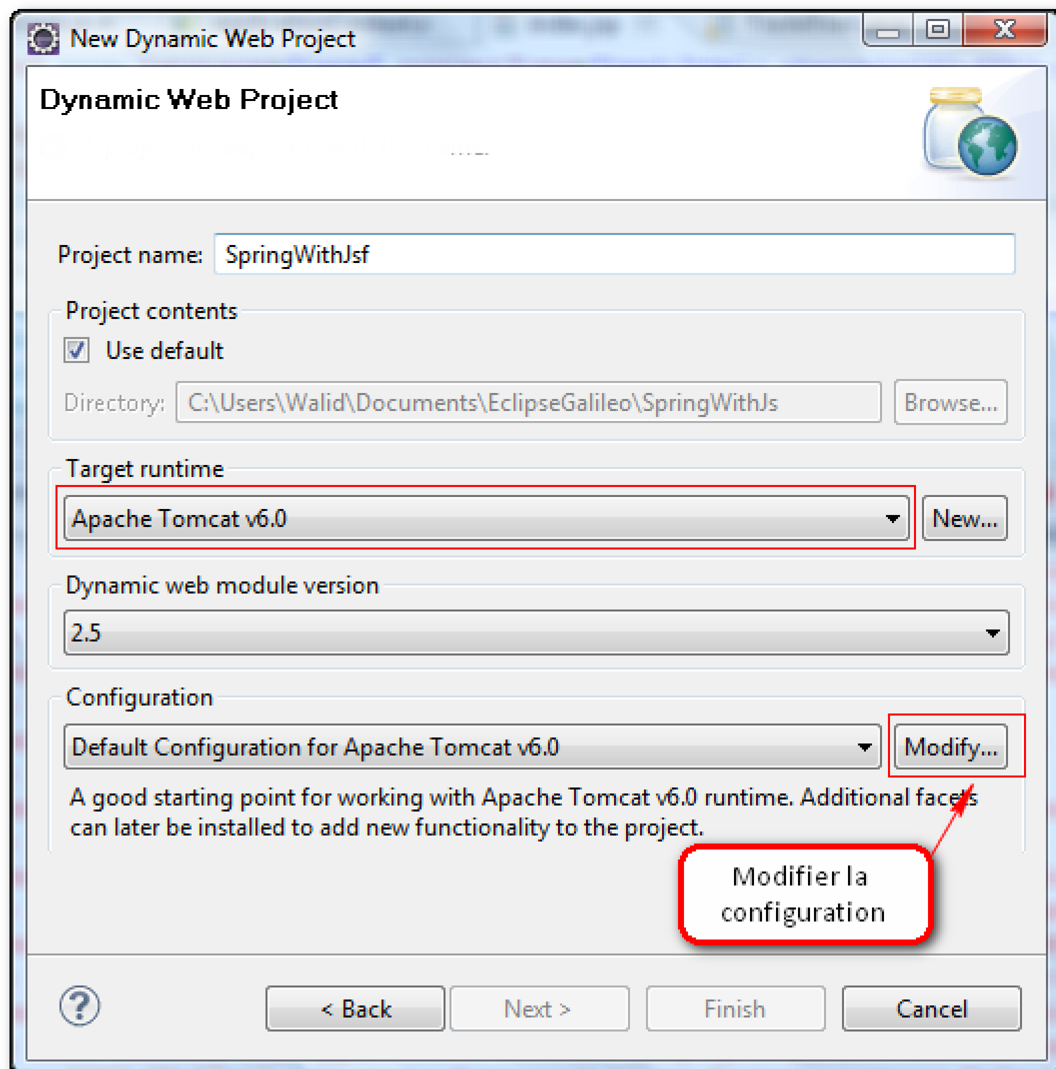
Dans cet exemple nous allons utiliser **Spring** pour gérer les dépendances des backing beans **JSF**. Nous supposons que vous êtes déjà familier avec JSF! Voici les étapes à suivre :

Création d'un projet web JSF de nature Spring :

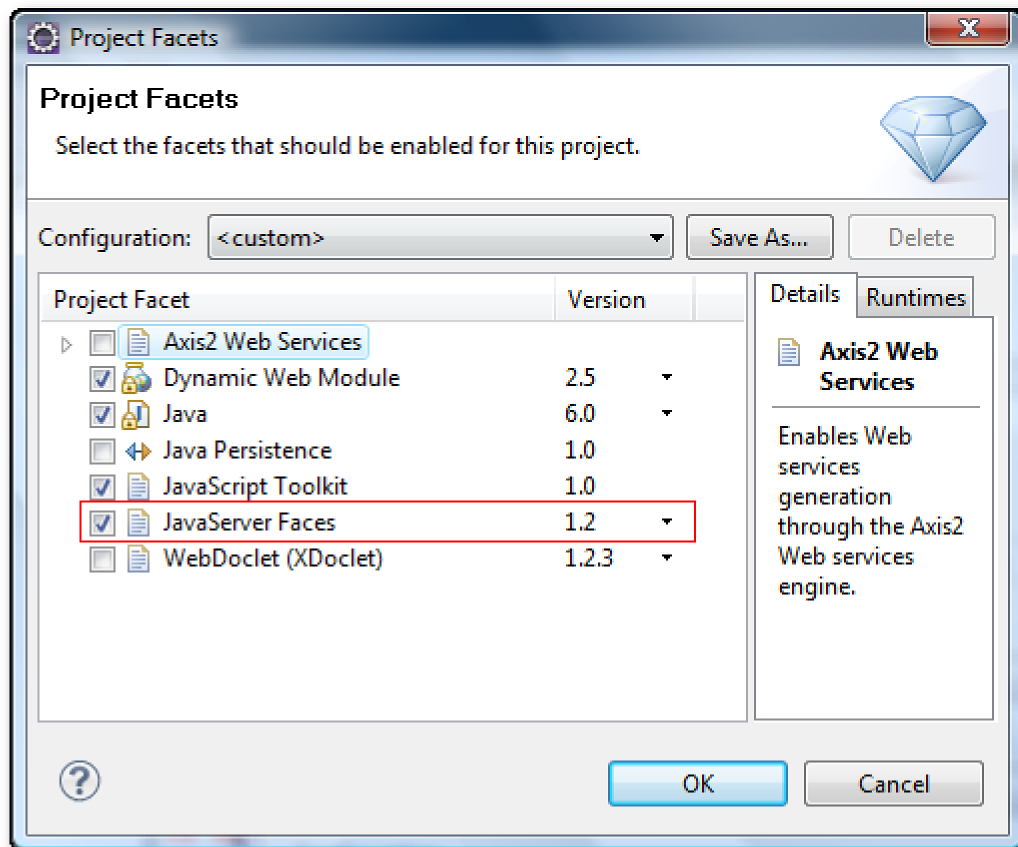
Créer tout d'abord un nouveau projet web dynamique :



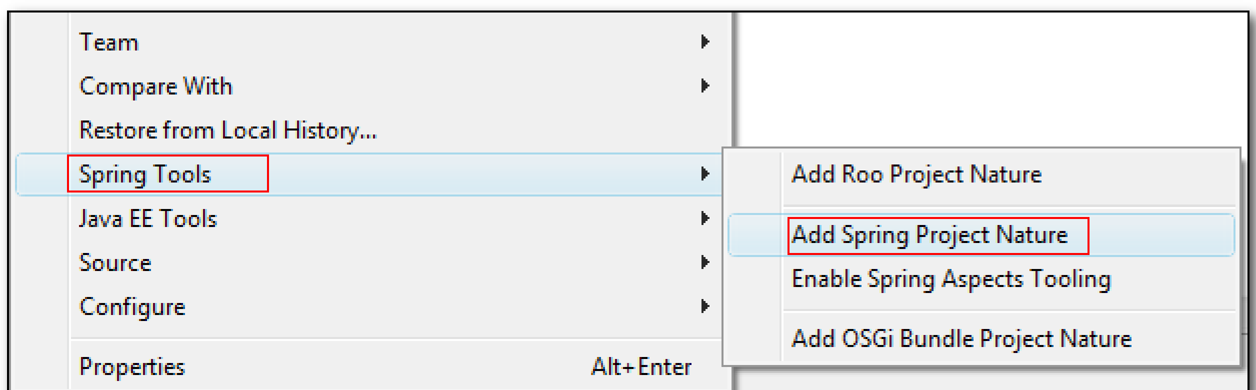
Ensuite, choisir le serveur ou le conteneur web, où votre application sera déployée. Vous devez aussi configurer l'application, en cliquant sur le bouton « **Modify**… » comme indiquer ci-dessous :



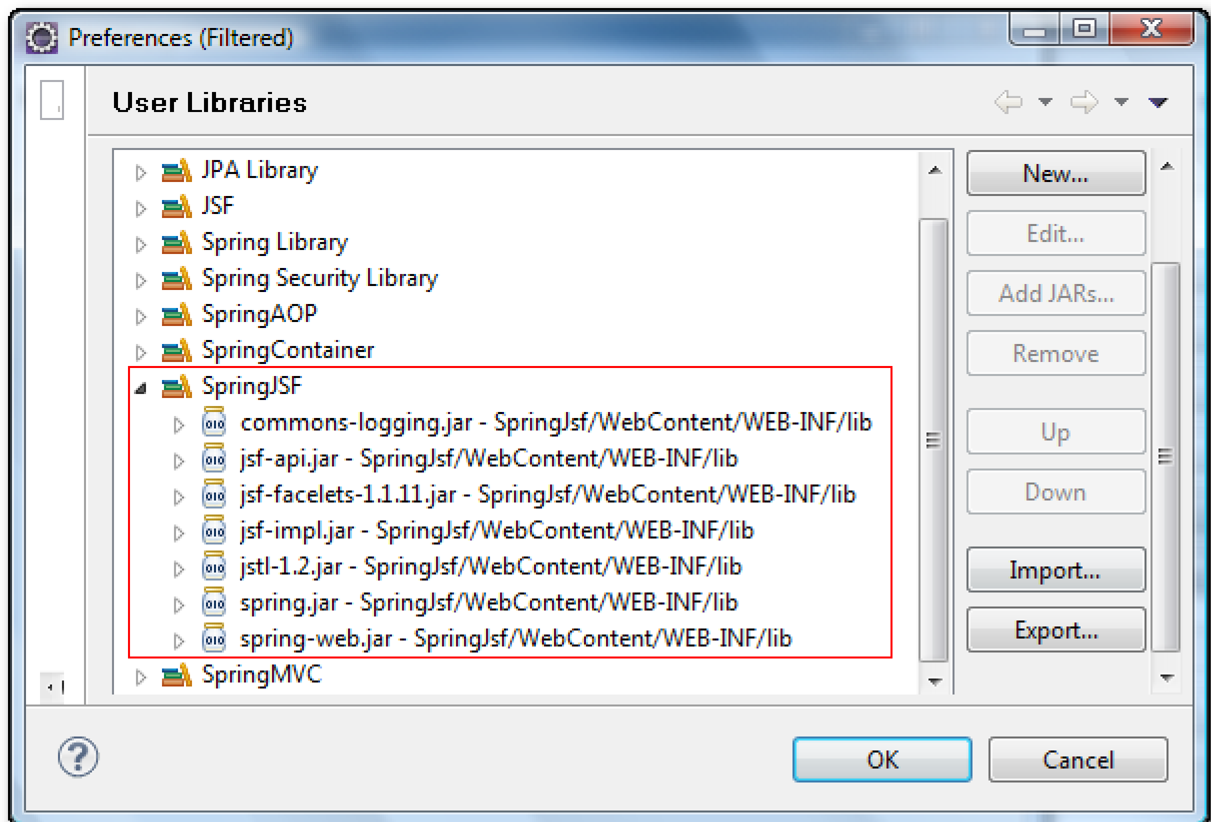
Ajouter JavaServer Faces à votre application web:



Ajouter la nature **Spring** à votre application :



Ajouter les jars suivants à votre application :



Configuration de web.xml

Ajouter dans l e descripteur de déploiement « web.xml » un **ContextLoaderListener** qui permettra de charger un **WebApplicationContext** à partir du fichier de définition des beans Spring, déclaré dans la variable de contexte **contextConfigLocation** :

```
<!-- Configuration Spring -->
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>

<listener>
      <listener-class>
            org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>

<listener>
      <listener-class>
            org.springframework.web.context.request.RequestContextListener
      </listener-class>
</listener>
<!-- Fin Configuration Spring -->
```

Configuration de faces-config.xml

La plus simple façon d'intégrer la couche métier de **Spring** avec la couche présentation **JSF**, est d'utiliser la classe **DelegatingVariableResolver**. Donc, ajouter cette déclaration dans le fichier de configuration de JSF **« faces-config.xml »** :

```
<application>
	<variable-resolver>
		org.springframework.web.jsf.DelegatingVariableResolver
	</variable-resolver>
</application>
```

Création de la classe Etudiant

Créer la classe étudiant comme suit :

```
package org.insat.springjsf.bean;

public class Etudiant {

	private String nom;
	private String prenom;
	private String filiere;

	public String getNom() {
		return nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public String getFiliere() {
		return filiere;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}

	public void setFiliere(String filiere) {
		this.filiere = filiere;
	}
}
```

Création de applicationContext.xml

Créer un fichier de définition des beans **Spring «applicationContext.xml »**, qui permettra l'instanciation d'un étudiant (ajouter le namespace Property lors de la création de ce fichier) :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

      <bean id="etudiant" class="org.insat.springjsf.bean.Etudiant"
      p:nom="Mohamed" p:prenom="Ali" p:filiere="Génie Logiciel" />

</beans>
```

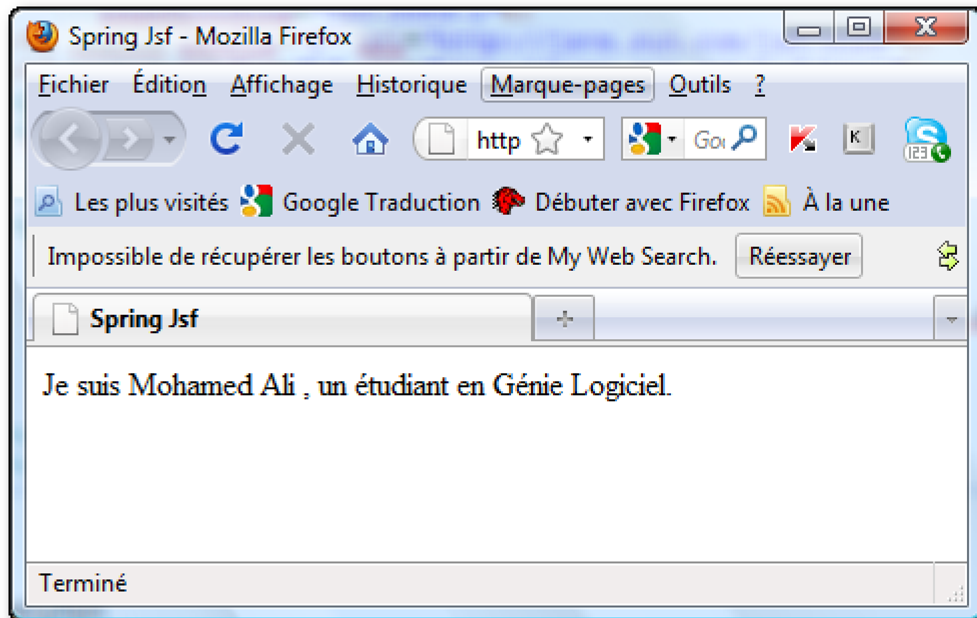
Création d'une page web

Créer une page web qui permettra l'accès aux données instanciés par Spring, en tant que ManagedBean JSF :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<html>
<head>
      <title>Spring Jsf</title>
</head>
<body>
      <f:view>
            Je suis
            <h:outputText value=" #{etudiant.nom} #{etudiant.prenom}" />
            , un étudiant en
            <h:outputText value=" #{etudiant.filiere}" />.
      </f:view>
</body>
</html>
```

Exécution

L'exécution de cette application résulte :



Atelier 7:
Spring Security

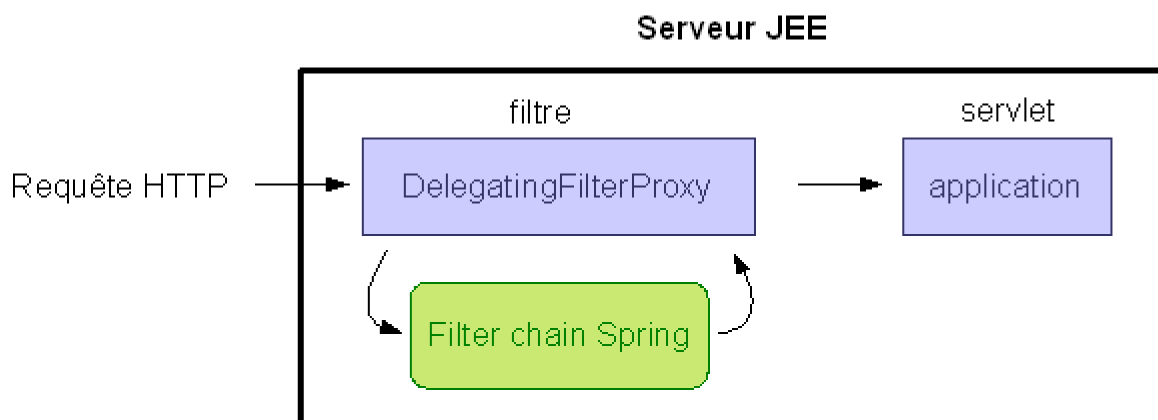
Atelier 7 - Spring Security

Principe

Spring Security permet de gérer l'accès aux ressources d'une application Java. Ces ressources peuvent être des pages web, mais aussi des objets de services métier. Toute ressource sollicitée par un appelant est rendue accessible si, d'une part, l'appelant s'est identifié, et si d'autre part, il possède les droits nécessaires (des rôles dans le vocabulaire Spring Security).

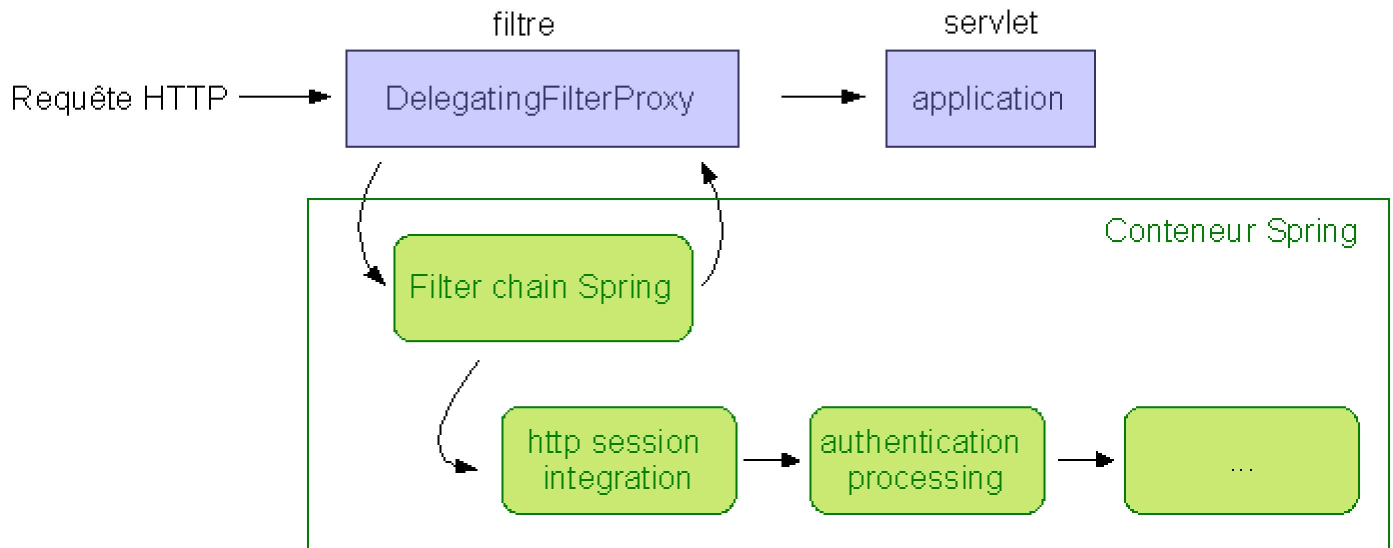
Le module *Acegi* a été rebaptisé en version 2 par *Spring Security*. Outre ce renommage, cette nouvelle version apporte des schémas XML qui simplifient considérablement la configuration.

Les requêtes HTTP sont interceptées par un filtre de servlet qui délègue à un bean Spring les traitements de vérification d'accès aux pages web.



Ce bean met en œuvre une chaîne de filtres. Chacun des filtres est un bean auquel est attribué une tâche précise :

- Intégration dans la session HTTP des informations de sécurité contenues dans la requête
- Vérification de l'identité de l'appelant et affichage d'une invite de connexion si nécessaire
- Vérification des droits d'accès à la ressource sollicitée
- ...



Certains filtres sont obligatoires, d'autres optionnels. La chaîne de filtres est largement configurable, ce qui permet de personnaliser au mieux la gestion de la sécurité dans les applications web. Spring Security offre ainsi les fonctionnalités suivantes :

- Authentification anonyme
- Fonction *Remember Me*
- Gestion NTLM
- Intégration avec un serveur LDAP ou un serveur CAS
- Gestion des certificats X509

Exemple d'utilisation

Dans cette partie vous allez créer pas à pas une application web et la sécuriser avec Spring Security.

- Créez dans eclipse un projet web dynamique 'SpringSecurity'
- Ajoutez des capacités Spring à votre projet (cliquez droit sur le projet puis Spring Tools/Add Spring Nature)

- Créez une page d'accueil index.htm :

```
<html>
<head>
<title>Spring Security</title>
</head>
<body>
<h1>Ceci est une page sécurisée</h1>
</body>
</html>
```

Ajouter les jars suivants à votre application :

Preferences (Filtered)

User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

- Jboss 5
- JPA Library
- Spring Library
- Spring Security Library
  - commons-codec.jar - SpringSecurity/WebContent/WEB-INF/lib
  - commons-logging.jar - SpringSecurity/WebContent/WEB-INF/lib
  - log4j-1.2.15.jar - SpringSecurity/WebContent/WEB-INF/lib
  - mysql-connector-java-5.1.9.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring-security-acl-2.0.4.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring-security-core-2.0.4.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring-security-core-tiger-2.0.4.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring-security-taglibs-2.0.4.jar - SpringSecurity/WebContent/WEB-INF/lib
  - spring-web.jar - SpringSecurity/WebContent/WEB-INF/lib
- SpringAOP

New... Edit... Add JARs... Remove Up Down Import... Export...

OK Cancel

Configuration du web.xml

Imposez la page index.htm comme étant la page d'accueil :

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

Déclarez l'intercepteur de sécurité. Pour cela, ajoutez le contenu suivant dans le fichier web.xml :

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
          org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

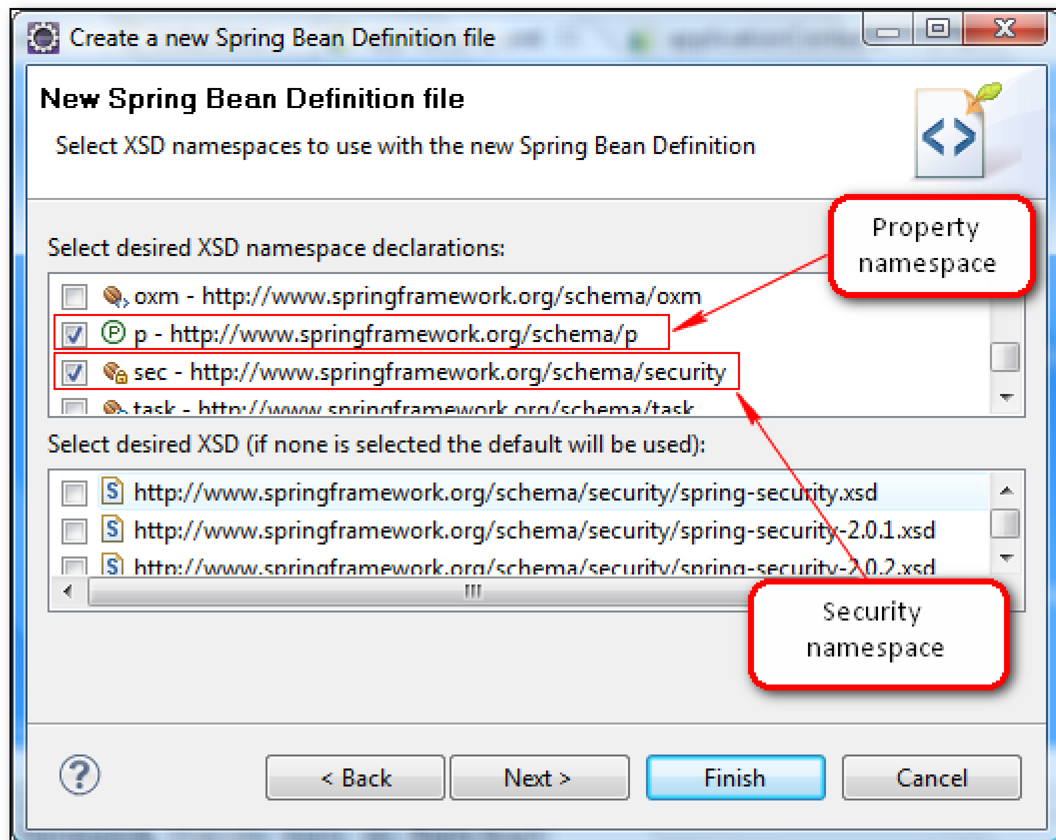
Configurez le chargeur de définition des beans : **ContextLoaderListener** :

```
<listener>
    <listener-class>
          org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/app-security.xml</param-value>
</context-param>
```

Le fichier web.xml est désormais bien configuré.

Configuration de app-security.xml

Créer le fichier de définition des beans **\WEB-INF\app-security.xml**. Lors de la création de ce fichier, sélectionner le namespace security et property :



Le fichier de définition des beans **app-security.xml** sera :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sec="http://www.springframework.org/schema/security"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-
2.0.4.xsd">

</beans>
```

Une configuration de base

Pour une configuration chaîne de filtres minimale, nous modifions le fichier **app-security.xml** comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sec="http://www.springframework.org/schema/security"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-2.0.4.xsd">


      <sec:authentication-provider>
            <sec:user-service>
                  <sec:user name="admin" password="admin" authorities="ROLE_ADMIN,ROLE_USER"
/>
                  <sec:user name="guest" password="guest" authorities="ROLE_USER" />
            </sec:user-service>
      </sec:authentication-provider>

      <sec:http auto-config="true">
            <sec:intercept-url pattern="/**" access="ROLE_ADMIN" />
            <sec:http-basic />
      </sec:http>

</beans>
```

Le volume de cette configuration est bien moindre, bien que cet exemple est complet et bien fonctionnel. Voici l'explication des éléments de cette configuration :

- La base de données utilisateur est définie dans le fichier de configuration. Pour chaque utilisateur, on spécifie l'identifiant, le mot de passe et les rôles qui lui sont attribués avec l'élément **`<sec:user />`**.
- On définit aussi les règles d'accès aux pages de l'application à l'aide de l'élément **`<sec:intercept-url />`**. Il s'agit simplement d'associer un pattern d'URL à un ou plusieurs rôles (séparation avec une virgule). A noter que les patterns d'URL sont traités dans l'ordre de déclaration : si /** était déclaré en premier, les autres patterns ne seraient pas vérifiés.
- Le dernier élément **`<sec:http-basic />`**permet d'afficher une invite de connexion avec une **popup** pour demander l'identification de l'utilisateur.

Configuration avec JDBC provider

Les informations d'authentification peuvent être stockées dans une base de données:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sec="http://www.springframework.org/schema/security"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-2.0.4.xsd">

      <bean id="dataSource"
            class="org.springframework.jdbc.datasource.DriverManagerDataSource"
                  p:driverClassName="com.mysql.jdbc.Driver"
                  p:url="jdbc:mysql://localhost:3306/libertysoft"
                  p:username="root" p:password="" />

      <sec:authentication-provider>
            <sec:jdbc-user-service data-source-ref="dataSource"
                  users-by-username-query="SELECT MAILMEMBRE, PASSWORDMEMBRE,
      'true' as enabled FROM membre WHERE MAILMEMBRE = ?"
                  authorities-by-username-query="SELECT MAILMEMBRE,ROLE
                  as authorities FROM rolemembre WHERE MAILMEMBRE = ?" />
      </sec:authentication-provider>

      <sec:http auto-config="true">
            <sec:intercept-url pattern="/**" access="ROLE_ADMIN, ROLE_MABABA" />
            <sec:http-basic />
      </sec:http>

</beans>
```

L'attribut `data-source-ref` permet de référencer un bean Spring de type DataSource.

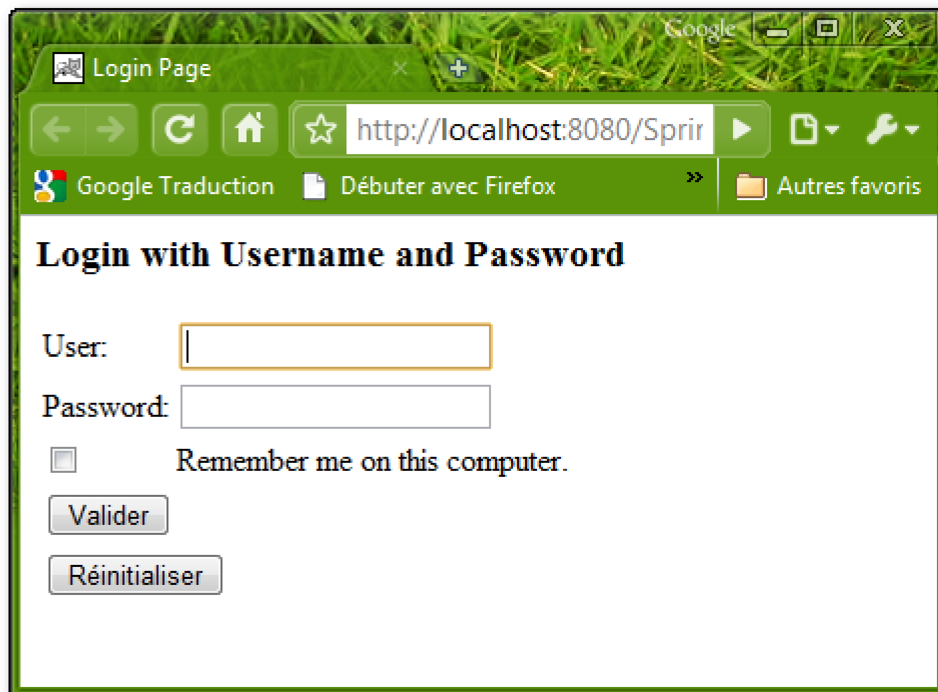
Sachant qu'on dispose d'une base de données Mysql nommée **libertysoft** et qui comporte une table **membre** pour les utilisateurs, une table **role** pour la gestion des rôles et une table **rolemembre**.

Si vous ne spécifiez pas les attributs `users-by-username-query` et `authorities-by-username-query`, par défaut Spring utilise une classe DAO qui respecte le schéma SQL suivant :

```
CREATE TABLE users (
  username VARCHAR(50) NOT NULL PRIMARY KEY,
  password VARCHAR(50) NOT NULL,
  enabled BIT NOT NULL
);

CREATE TABLE authorities (
  username VARCHAR(50) NOT NULL,
  authority VARCHAR(50) NOT NULL
);

ALTER TABLE authorities ADD CONSTRAINT fk_authorities_users foreign key
(username) REFERENCES users(username);
```

Authentification par formulaire

Si vous voulez activer l'authentification par formulaire, il suffit de remplacer l'élément `<sec:http-basic />` par l'élément `<sec:form-login />`. Spring Security génère une page de formulaire par défaut :



Même il est possible de spécifier deux pages d'authentification et d'échec personnalisées à l'aide des attributs `login-page` et `authentication-failure-url`:

```
<sec:http auto-config="true">
      <sec:intercept-url pattern="/**" access="ROLE_ADMIN" />
      <sec:form-login login-page="/login.jsp"
            authentication-failure-url="/failure.jsp" />
</sec:http>
```

Par ailleurs, la page de login doit respecter certaines règles :

- Action : j_spring_security_check
- Nom input identifiant : j_username
- Nom input mot de passe : j_password

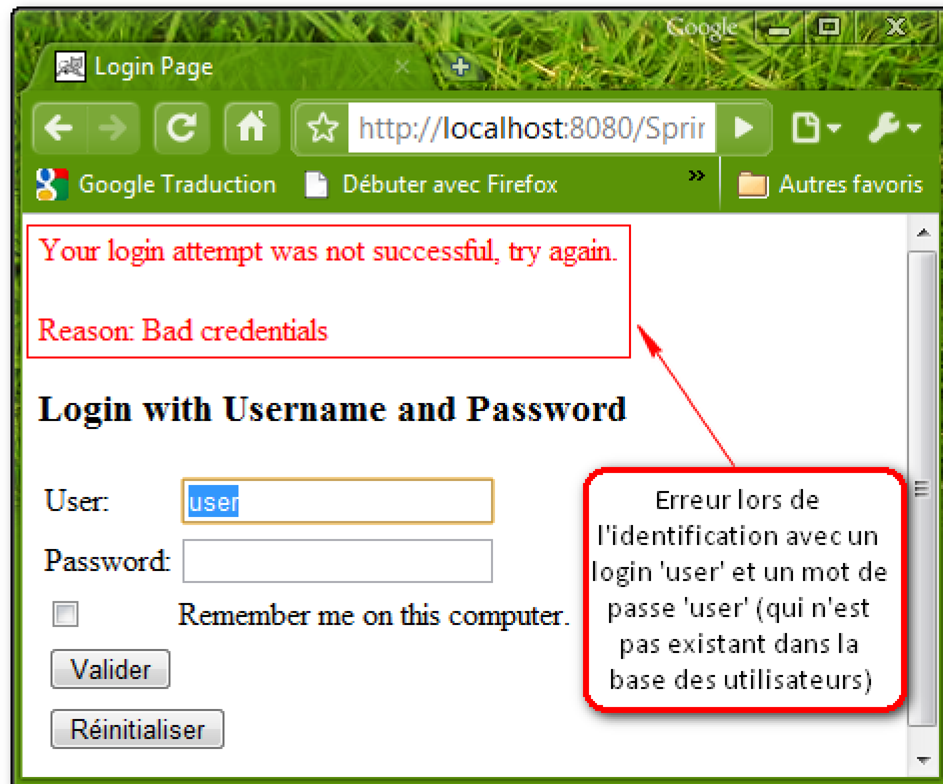
```
<form method="post" action="j_spring_security_check">
      Identifiant :
      <input name="j_username" value="" type="text" />
      Mot de passe :
      <input name="j_password" type="password" />
      <input value="Valider" type="submit" />
</form>
```

Test de l'application

Dans notre exemple, toutes les pages sont filtrées. La demande d'identification se produit quelle que soit la page demandée. Une fois l'utilisateur identifié, les informations de sécurité le concernant sont stockées en session HTTP dans le security context. L'identification n'est par conséquent pas demandée une nouvelle fois lorsque l'on navigue vers d'autres pages.

Si l'utilisateur tente d'accéder une ressource non autorisée (par exemple page index.html pour l'utilisateur guest), Spring Security renvoie une erreur 403 au navigateur web :

Si l'identification de l'utilisateur est incorrecte (identifiant ou mot de passe), Spring Security renvoie une erreur au navigateur web :



Si l'identification de l'utilisateur est correcte et l'accès à la ressource est autorisée :

Atelier 8: Spring .NET

Atelier 8 - Spring .NET

Principe

Suite au succès de la version Java de Spring, ce n'était qu'une question de temps avant que la version .NET de Spring est arrivée.

Spring.NET est un Framework qui fournit un support complet visant l'infrastructure des applications .NET. Le cœur de Spring.NET est le concept de dépendance d'injection et plus généralement l'inversion de contrôle.

En effet, Spring.NET fournit un ensemble d'outils qui aident à respecter les bonnes pratiques (Injection de dépendance, Programmation Orientée Aspect…).

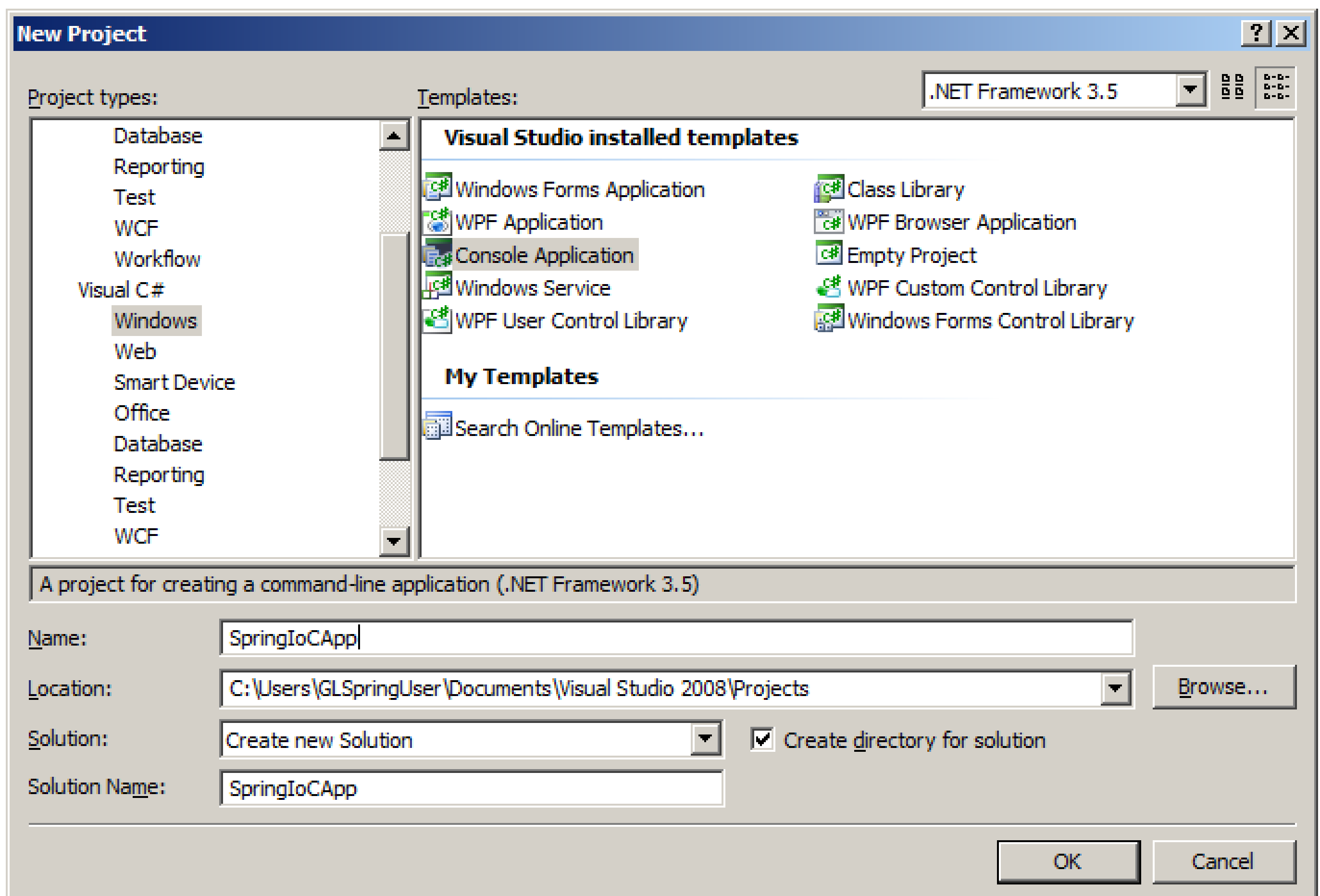
Exemple d'utilisation

A travers cet atelier, nous allons présenter les principes de base d'injection de dépendance (Dependency Injection) de Spring.NET à travers une simple application console.

Nous allons reproduire le même exemple décrit dans l'atelier « Inversion de Contrôle (IoC) » réalisé en java.

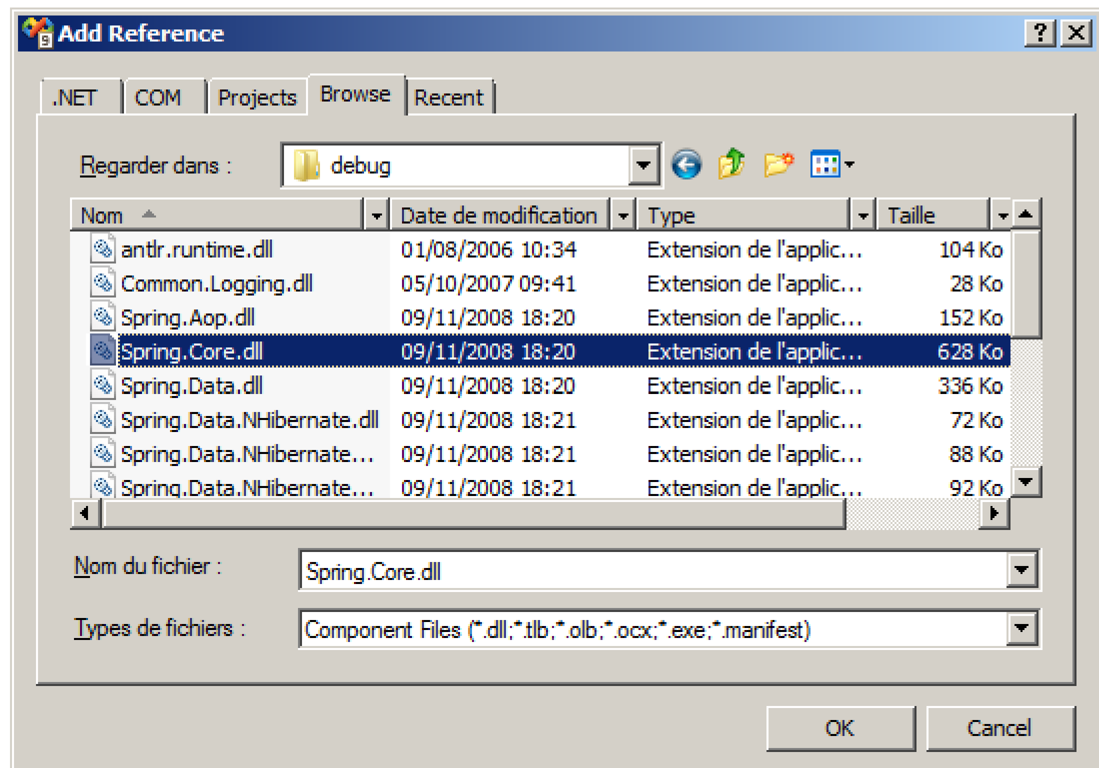
Création du projet

Créez un nouveau projet de type « Console Application » :



Ajoutez la référence de Spring.NET à votre projet. Pour cet exemple, nous avons besoin seulement de « Spring.Core.dll ». Les autres références peuvent être nécessaires dans des cas plus complexes.

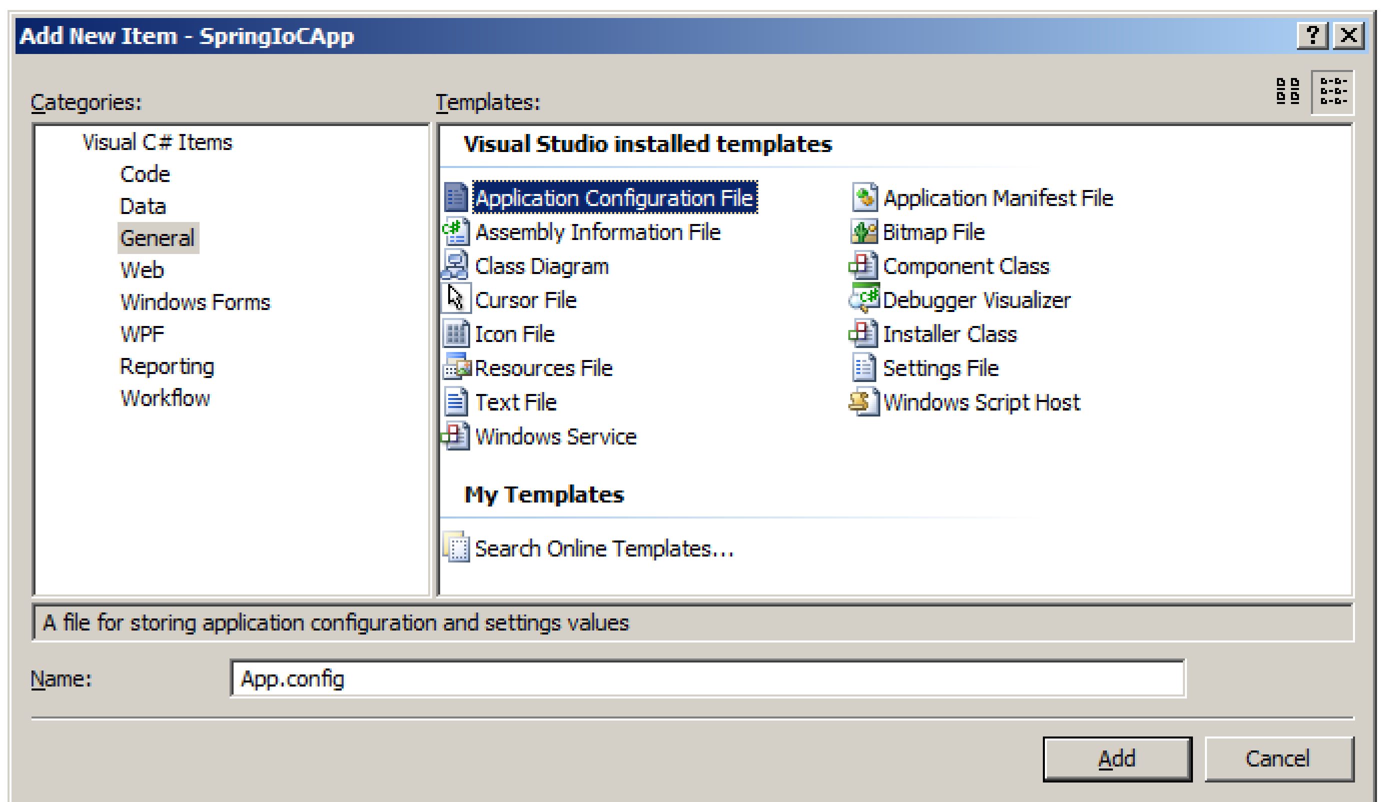
« Spring.Core.dll » se trouve sous le répertoire « ~\Spring.NET\Spring.NET\bin\net\2.0\debug »



Configuration de Spring.Net

Il existe plusieurs moyens d'obtenir une référence à une instance du conteneur Spring.NET : « IApplicationContext ». Pour cet exemple, nous allons utiliser une IApplicationContext instanciée à partir du fichier de configuration standard des applications .NET.

Commencez alors par ajouter le fichier de configuration à votre application :



Ajoutez le code suivant au fichier de configuration « *App.config* » (ou « *Web.config* » s'il s'agit d'une application web).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context"
               type="Spring.Context.Support.ContextHandler,
                     Spring.Core" />
      <section name="objects"
               type="Spring.Context.Support.DefaultSectionHandler,
                     Spring.Core" />
    </sectionGroup>
  </configSections>

  <spring>
    <context>
      <resource uri="config://spring/objects" />
    </context>
    <objects xmlns="http://www.springframework.net">
      <description>Exemple simple sur l'utilisation de Spring IoC.</description>
    </objects>
  </spring>
</configuration>
```

Ceci présente configuration de base pour Spring.Net. A ce stade, vous pouvez commencer à ajouter les définitions des objets à l'intérieur du tag objects (<objects> </objects>).

Ajout des classes

Nous allons maintenant refaire le même travail que dans l'atelier IoC avec une implémentation basé sur Spring.NET.

Créez alors les classes suivantes :

Interface IFiliere :

```
using System;

namespace SpringIoCApp
{
    interface IFiliere
    {
        String Nom { get;   }
    }
}
```

Une filière est caractérisée par son nom.

Classes FiliereGL :

Cette classe implémente l'interface IFiliere.

```
using System;

namespace SpringIoCApp
{
    class FiliereGL : IFiliere
    {
        private String _nom;

        public string Nom
        {
            get { return _nom; }
            set { _nom = value; }
        }

    }
}
```

Classes Etudiant :

```
using System;

namespace SpringIoCApp
{
    class Etudiant
    {
        private IFiliere _filiere;

        public Etudiant()
        {
        }

        public Etudiant(IFiliere filiere)
        {
            _filiere = filiere;
        }

        public IFiliere Filiere
        {
            get
            {
                return _filiere;
            }
            set
            {
                _filiere = value;
            }
        }
    }
}
```

Un étudiant dispose d'une filière.

Définition des objets

Pour l'instant, aucun objet n'a été définit dans le fichier de configuration de l'application. Nous allons les définir maintenant.

Ajoutez alors l'extrait XML suivant à l'intérieur du tag <objects> du fichier « App.config ».

```
<object name="Etudiant"
        type="SpringIoCApp.Etudiant, SpringIoCApp"
        >
  <constructor-arg name="filiere" ref="FiliereGL" />
</object>

<object name="FiliereGL"
      type="SpringIoCApp.FiliereGL, SpringIoCApp">
  <property name="nom" value="Génie Logiciel"/>
</object>
```

Il est à noter que le tag <object> de Spring.NET est l'équivalent du tag <bean> de Spring Java.

Notez aussi que nous avons spécifié le nom complet de la classe Etudiant (« SpringIoCApp.Etudiant ») au sein de l'attribut type de la définition de l'objet.

Setter Injection

Le code suivant permet d'injecter le nom de la filière dans l'instance « FiliereGL » identifié par l'ID « FiliereGL ».

```
<property name="nom" value="Génie Logiciels"/>
```

Constructor Injection

Enfin, pour définir la filière de l'étudiant, nous avons définis la référence à l'instance « FiliereGL » comme un argument du constructeur de l'objet Etudiant. Ainsi, lorsque l'objet Etudiant est initialisé, la filière sera initialisée dans le constructeur.

```
<constructor-arg name="filiere" ref="FiliereGL" />
```

Récupération de l'objet

Exécutez le code suivant pour récupérer l'objet à partir du contexte Spring.Net:

```
using System;
using Spring.Context;
using Spring.Context.Support;

namespace SpringIoCApp
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            Etudiant etudiant = (Etudiant)ctx.GetObject("Etudiant");
            Console.Write("Filière : " + etudiant.Filiere.Nom );
        }
    }
}
```

Le rôle de ce code est de récupérer le contexte de l'application Spring.Net et demander ensuite une instance de l'objet « Etudiant ». Ce morceau de code ne connaît que le nom de l'objet et l'interface à utiliser. Il ne connait pas quelle classe est implémentée (dans notre cas c'est la classe FiliereGL) et il n'a pas à s'inquiéter sur la façon d'injecter le bon nom de filière dans l'objet car Spring.Net traite tout ça.

Exécution