Objectifs

- Déclarer dans le conteneur Spring des javabeans indépendants
- Comprendre le chargement du conteneur Spring
- Mettre en œuvre l'injection de dépendances
- Utiliser les vues STS, dont la vue graphe.

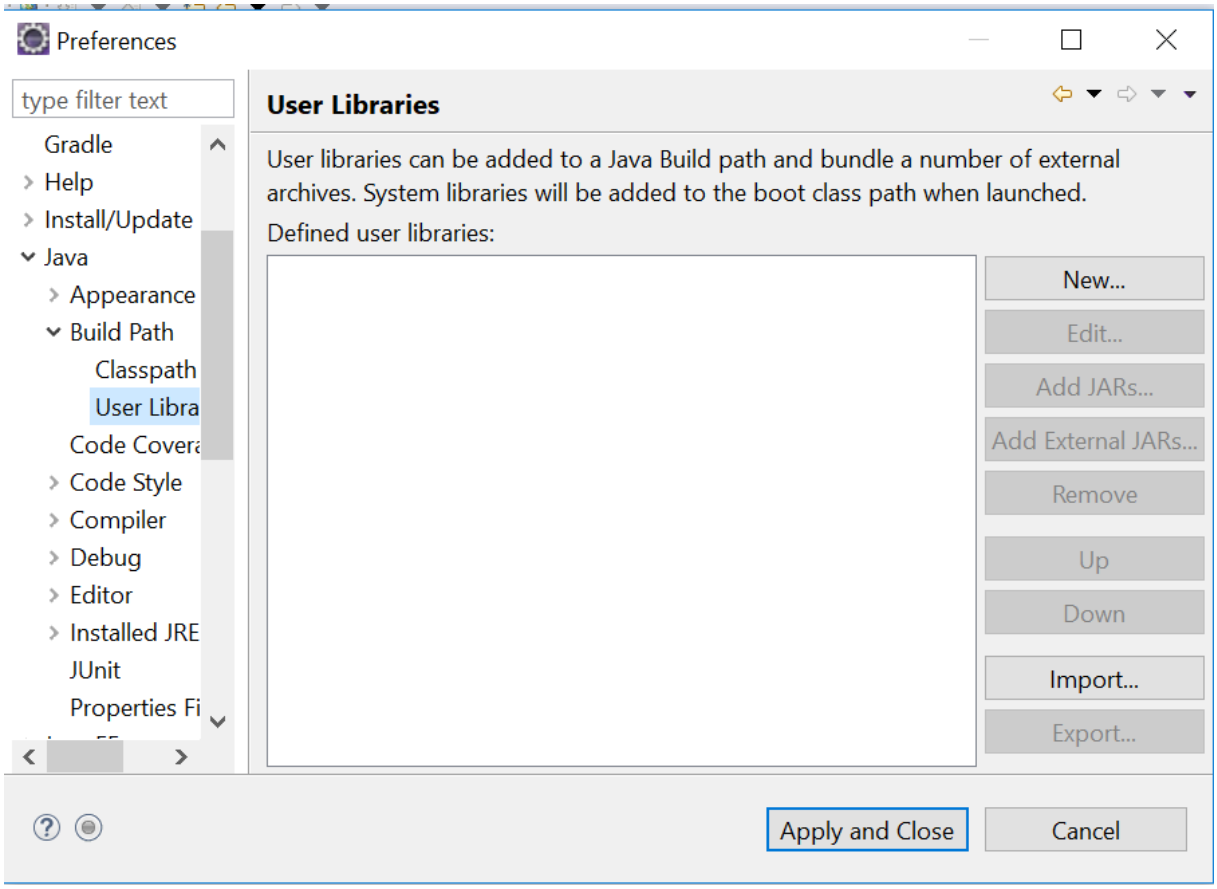
Durée

1h.

Partie 1 : Configuration des librairies du projet spring

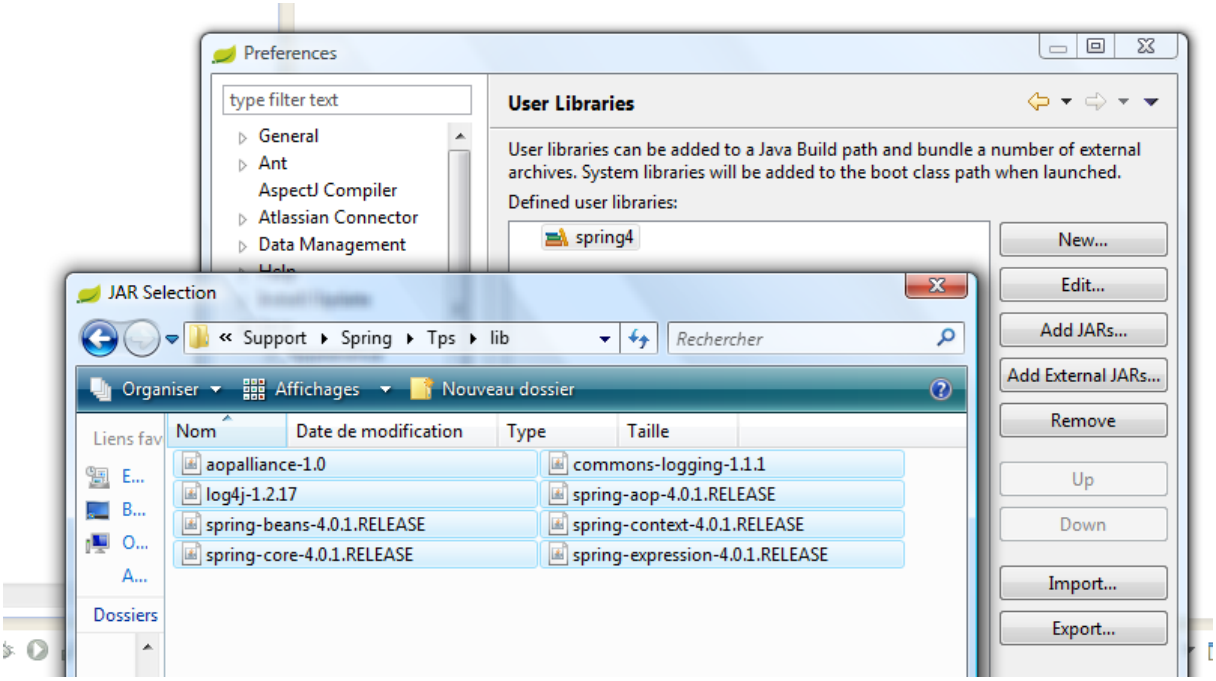
Ces dépendances se trouvent dans le **répertoire lib** fournit.

Dans Eclipse ➔ Windows ➔ Preferences ➔ Java ➔ Build Path ➔ User Librairies ➔ New ➔ Entrez le nom **spring4**



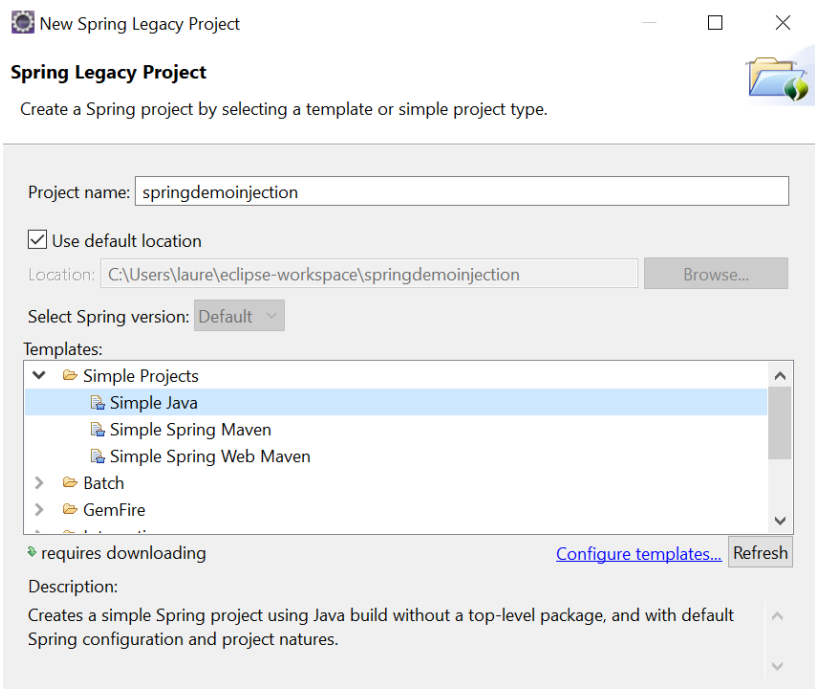
puis OK

Cliquer sur Add External Jars puis ajouter les librairies fournies puis OK

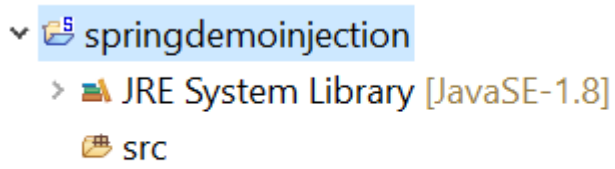


Partie 2 : création projet Spring

- File/New/Spring Legacy Project et sélectionner Simple Java



- Appuyez sur le bouton Finish. Voici ce qui apparaît dans Eclipse



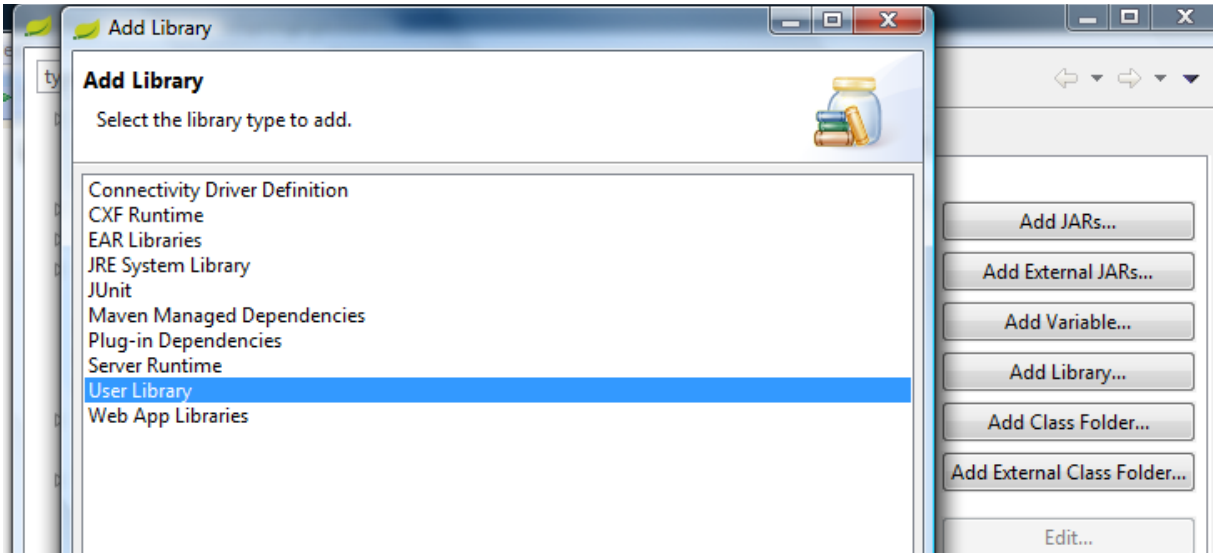
▸ **REMARQUE** : le 'S' en bleu à gauche du projet signifie que c'est un projet Spring. C'est la marque de projet Spring ou de projets ayant acquis des 'Capacités Spring'.

▸ **REMARQUE** : lorsque vous souhaitez donner à un projet non Spring des 'capacités Spring', cliquez droit sur le projet puis 'Spring Tools/Add Spring Project Nature'.

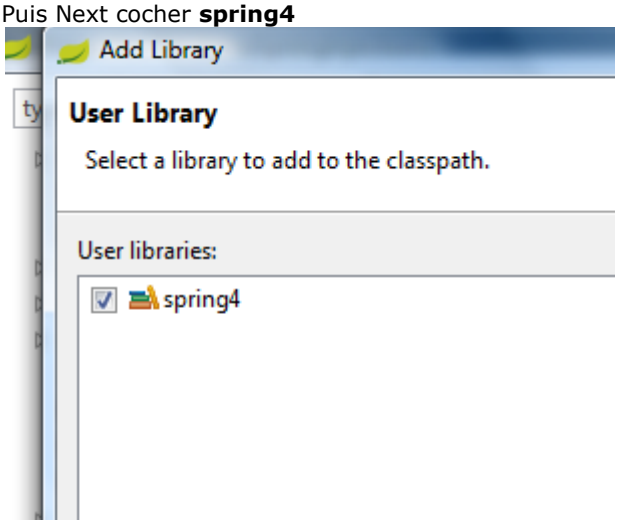
Configuration du build Path dans Eclipse

▸ Cliquez-droit sur le projet —> configure build path

▸ Dans l'écran qui apparaît, sélectionnez l'onglet 'Libraries' et cliquez sur le boutton 'Add library' qui permet d'ajouter des jars relatifs au projet, comme ceux que nous venons d'ajouter dans **'spring4'**.



Puis Next cocher **spring4**



Finish puis OK

configuration log4j

- Créer à la racine 'src' (en cliquant droit sur scr puis New/File) le fichier **log4j.xml**
- Ajoutez dans le fichier le contenu suivant.

```
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
	<appender name="console" class="org.apache.log4j.ConsoleAppender">
		<layout class="org.apache.log4j.SimpleLayout" />
	</appender>
	<root>
		<level value="info" />
		<appender-ref ref="console" />
	</root>
```

```
</log4j:configuration>
```

Partie 3 : codage des beans

Dans cette partie, vous créez deux Javabeans indépendants l'un de l'autre.

- Dans le projet 'demospringinjection', créez un package com.formation.spring.demo

Création du bean Developpeur

- Créez la classe Developpeur, comme ci-dessous

```
package com.formation.spring.demo;

public class Developpeur {

	private String nom;
	private int anneesExperience;


	//Get + set
}
```

- CONSEIL : faîtes générer les Getters / Setters : après la déclaration des attributs 'nom' et 'anneesExperience', cliquez droit sur le source puis '/Source/generate Getters and Setters' pour terminer la création de la classe.

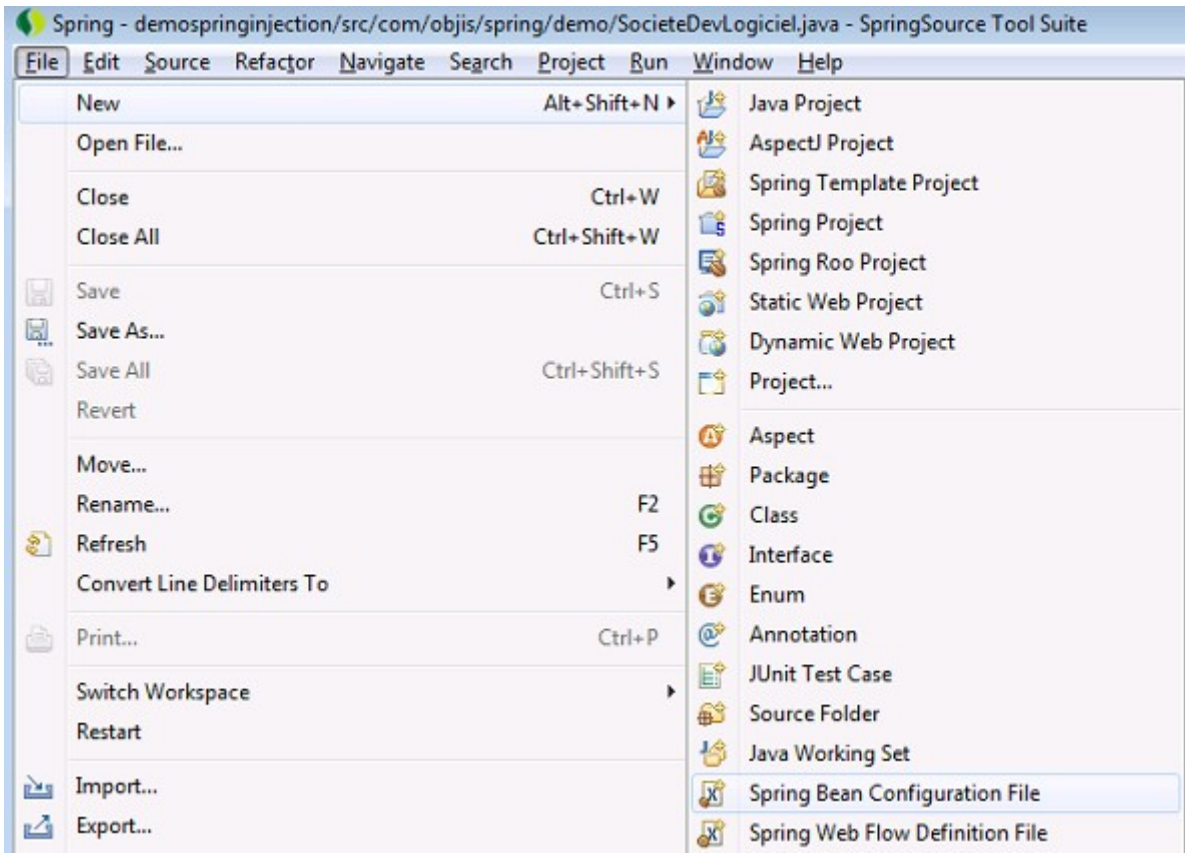
Création du bean SocieteDevLogiciel

- Créez la classe SocieteDevLogiciel, comme ci-dessous

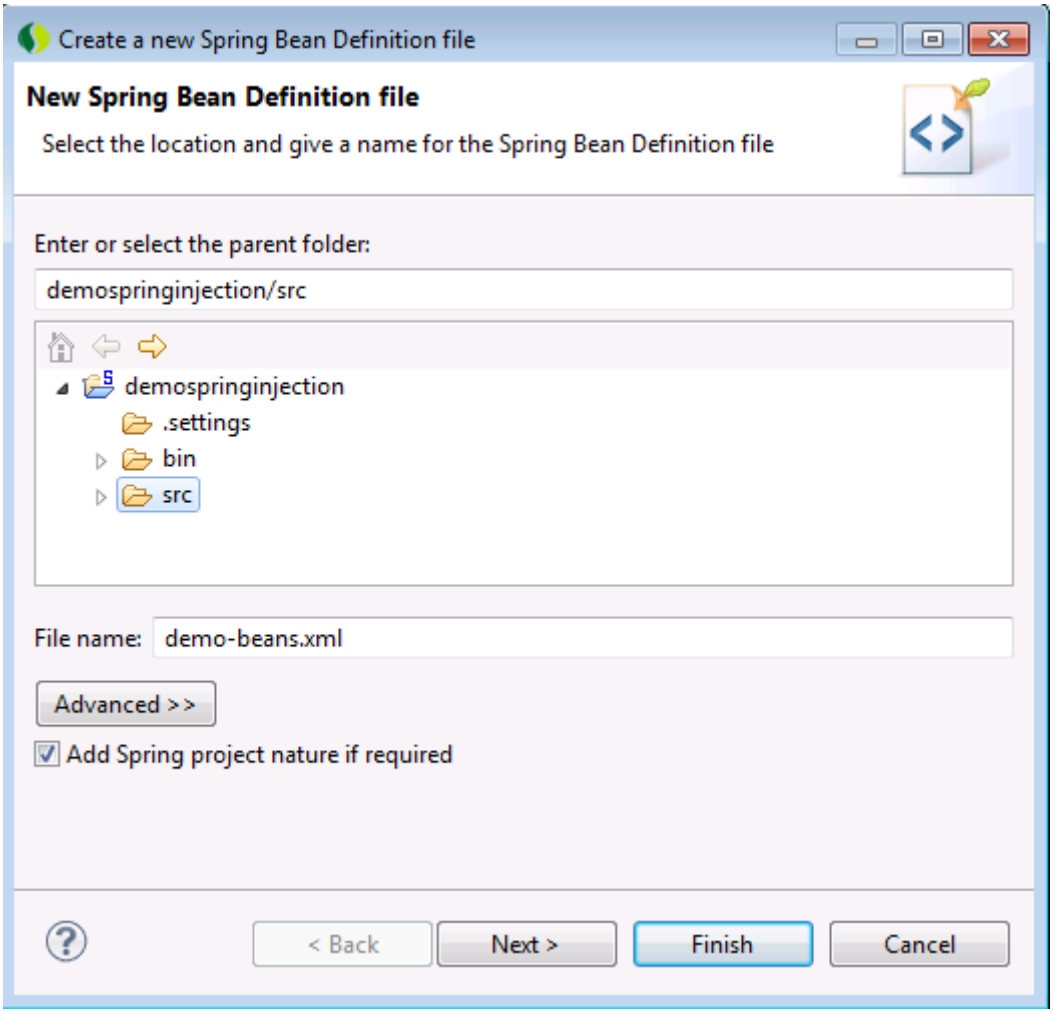
```java
package com.formation.spring.demo;

public class SocieteDevLogiciel {

	private Developpeur developpeur;
	private Developpeur chefDeveloppeur;

	public Developpeur getDeveloppeur() {
		return developpeur;
	}
	public void setDeveloppeur(Developpeur developpeur) {
		this.developpeur = developpeur;
	}
	public Developpeur getChefDeveloppeur() {
		return chefDeveloppeur;
	}
	public void setChefDeveloppeur(Developpeur chefDeveloppeur) {
		this.chefDeveloppeur = chefDeveloppeur;
	}
}
```

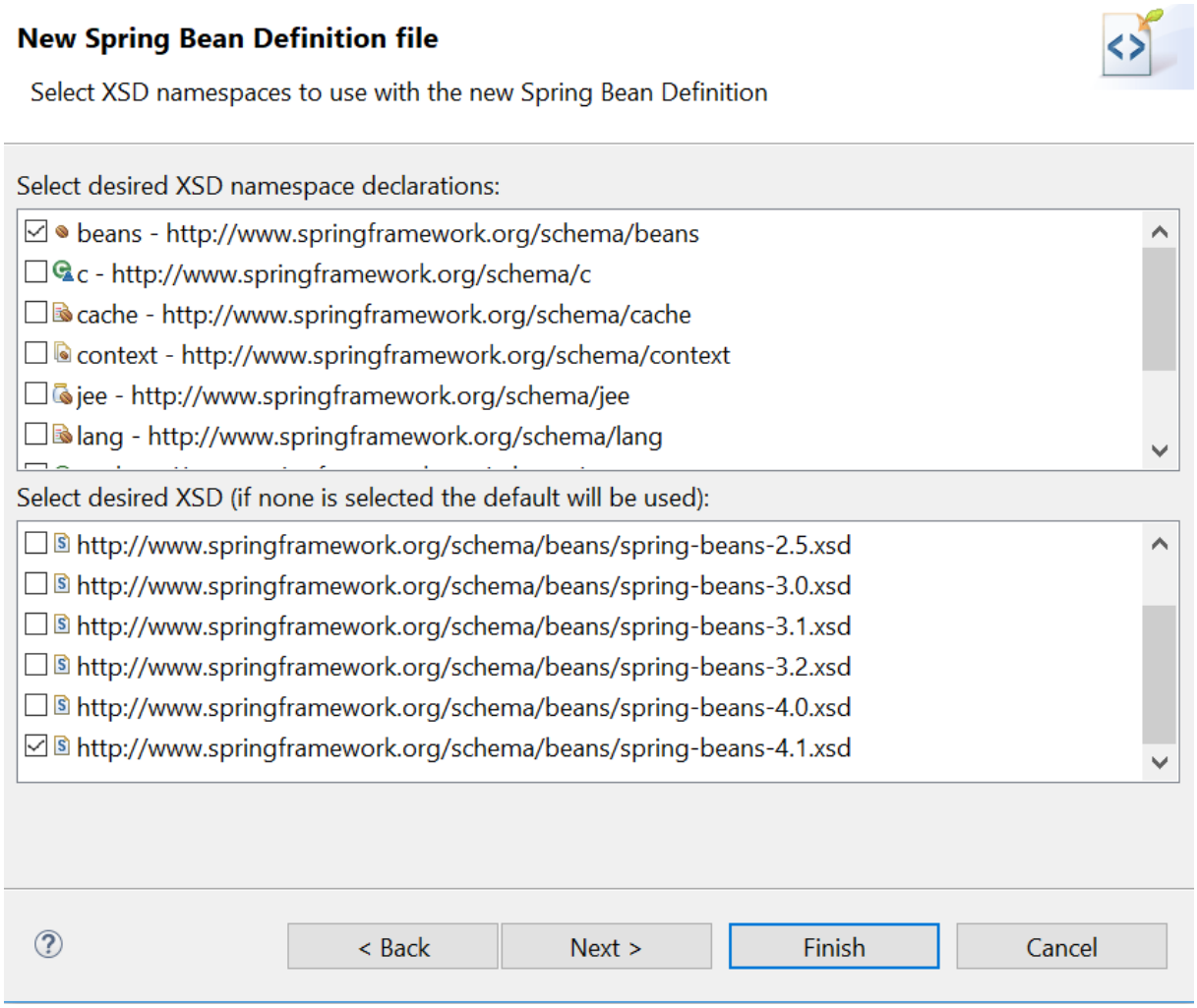
Partie 4 : Création fichier définitions de beans

- Menu : File/New/Spring Bean Configuration File



L'écran suivant apparaît :



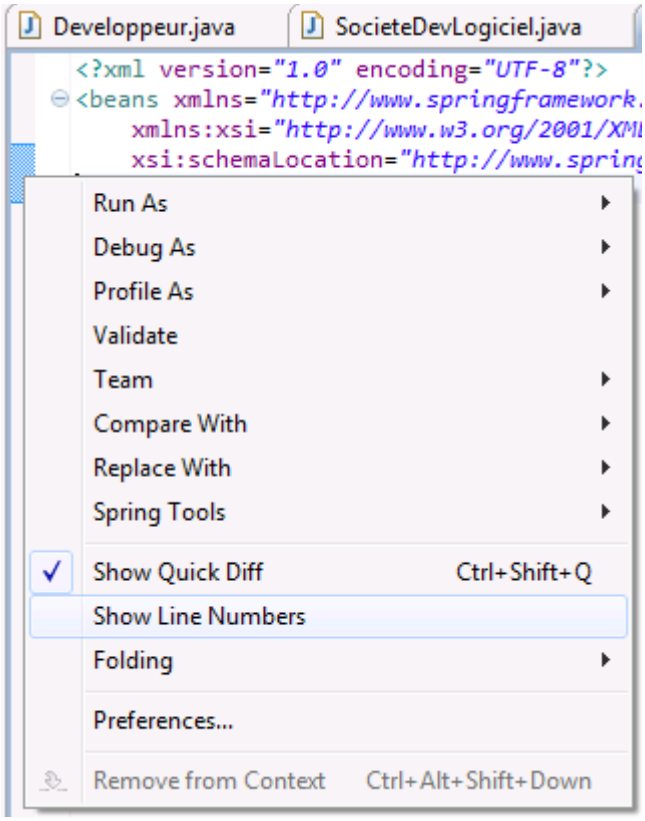


- Donnez le nom 'demo-beans.xml' au fichier de définition de beans. Puis cliquez sur Finish.

Eclipse génère alors le fichier avec comme contenu les lignes suivantes :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/be
5
6
7 </beans>
8
```

- Ajoutez des numéros de lignes au fichier (si elles n'apparaissent pas déjà) :



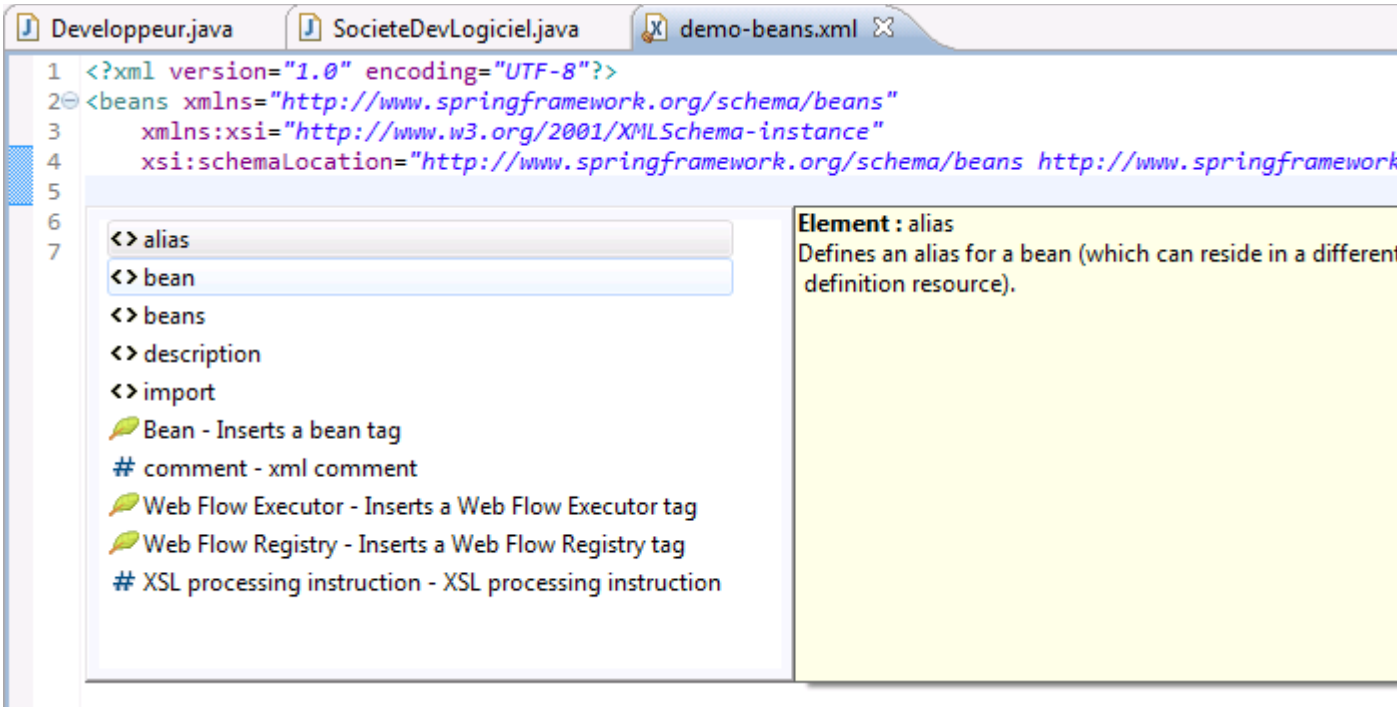
Reste désormais à ajouter la déclaration des deux beans codés plus haut : **Developpeur** et **SocieteDevLogiciel**

Partie 5 : Déclaration des beans dans spring

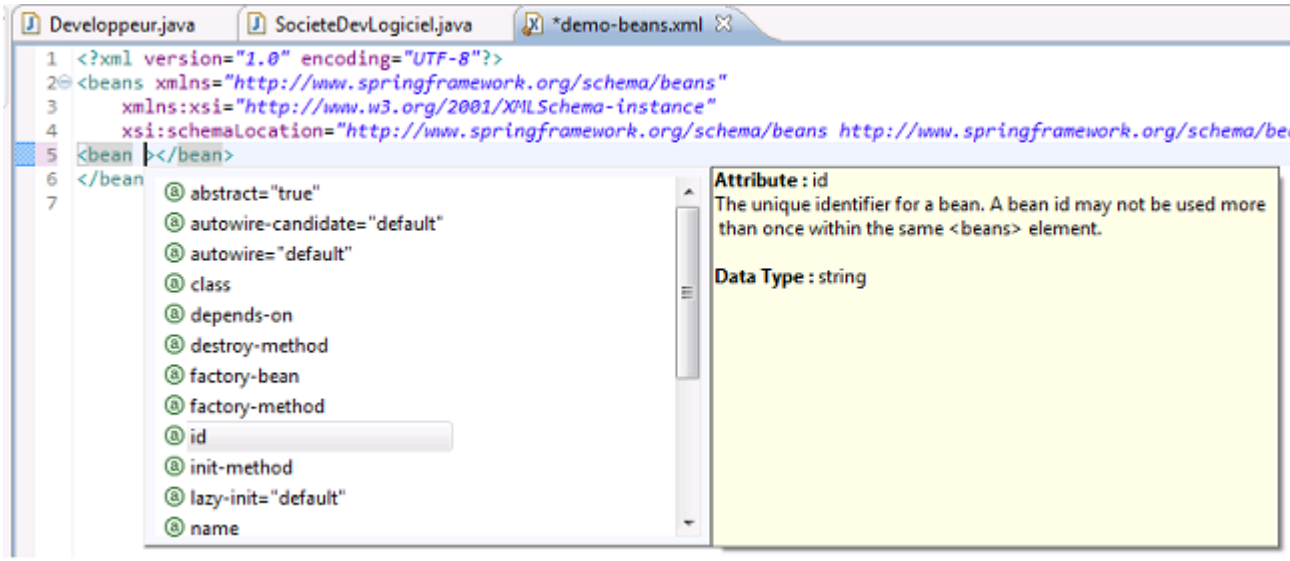
Editeur XML et complétion de code

- Déclarez un Développeur Olivier possédant 7 ans d'expériences.

Pour cela , suite à un CTRL + Espace dans la balise <beans> ,sélectionnez la balise <bean>



- Idem pour sélectionner l'attribut 'id'...



et l'attribut 'class' de la balise 'bean'

```
<bean id="developpeur" class="Dev" ></bean>
```

+ ctrl space pour autocompletion

- Injectez les propriétés 'nom' et 'anneesExpereince' de façon statique :

```
<bean id="developpeur" class="com.formation.spring.demo.Developpeur">
    <property name="nom" value="Ollivier"></property>
    <property name="anneesExperience" value="7"></property>
</bean>
```

Remarquez les 2 façons (attribut ou balise) de déclarer une propriété du bean.

```
<property name="anneesExperience">
    <value>7</value>
</property>
```

ça y est, la déclaration du bean Developpeur est terminée.

▸ **A VOUS DE JOUER** : Déclarez dans le même fichier de configuration demo-beans.xml un deuxième développeur nommé Franck, plus expérimenté qu'Olivier (10 ans d'expérience) et possédant dans le conteneur Spring un id='chefDeveloppeur'.

▸ **A VOUS DE JOUER** : Déclarez dans le même fichier de configuration demo-beans.xml une société de développement logicielle possédant les 2 développeurs déja déclarés, et possédant un id='societeDevLogiciel'. Utilisez l'attribut **'ref'** de la balise bean en pointant sur l'id de chacun des beans.

Solution :

```
<bean id="chefDeveloppeur" class="com.formation.spring.demo.Developpeur">
    <property name="nom" value="Franck"></property>
    <property name="anneesExperience" value="10"></property>
</bean>


<bean id="societeDevLogiciel" class="com.formation.spring.demo.SocieteDevLogiciel">
    <property name="chefDeveloppeur" ref="chefDeveloppeur"></property>
    <property name="developpeur" ref="developpeur"></property>
</bean>
```

Partie 6 : client

▸ Codez et analysez le contenu de la classe Principale 'DemoApp' suivante :

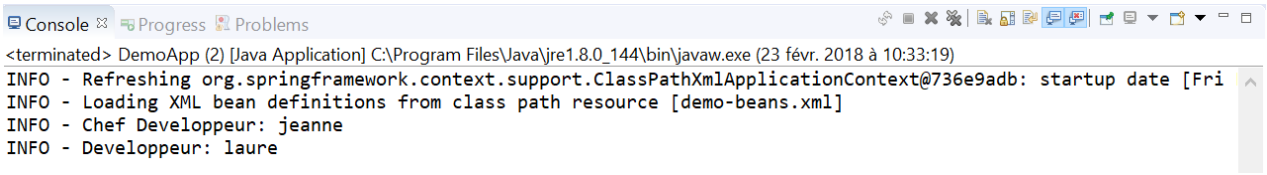
```
package com.formation.spring.demo.test;

import org.apache.log4j.Logger;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.formation.spring.demo.SocieteDevLogiciel;

public class DemoApp {
```

```
  private static final Logger logger= Logger.getLogger(DemoApp.class);

 public static void main(String[] args) {
   ApplicationContext  beanFactory = new ClassPathXmlApplicationContext("demo-beans.xml");

SocieteDevLogiciel societe= (SocieteDevLogiciel) beanFactory.getBean("esgi");

       logger.info("Chef Developpeur: "+societe.getChefDeveloppeur().getNom());
       logger.info("Developpeur: "+societe.getDeveloppeur().getNom());

  }
}
```

▸ Expliquez les lignes de code de la méthode main(). En particulier, mettre en évidence les 3 étapes suivantes :
— 1.Chargement du conteneur Spring
— 2.Récupération d'un bean du conteneur
— 3.Utilisation du bean avec injection de dépendances

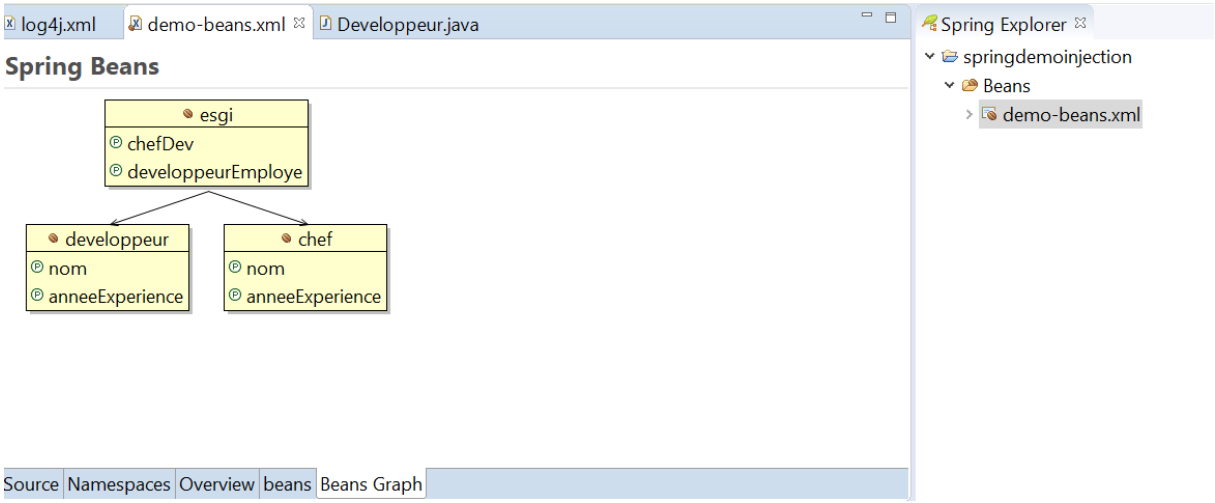
Résultat

- Après clic droit/Run as/ Java Application, vous obtenez sur la console ceci



Vues Spring

— Vue Beans Graph



Conclusion Dans ce tutoriel, vous avez pratiqué la mise en œuvre du concept clé N°1 de Spring :l'injection de dépendances.