# Atelier Framework Spring-02
# Injection de dépendances

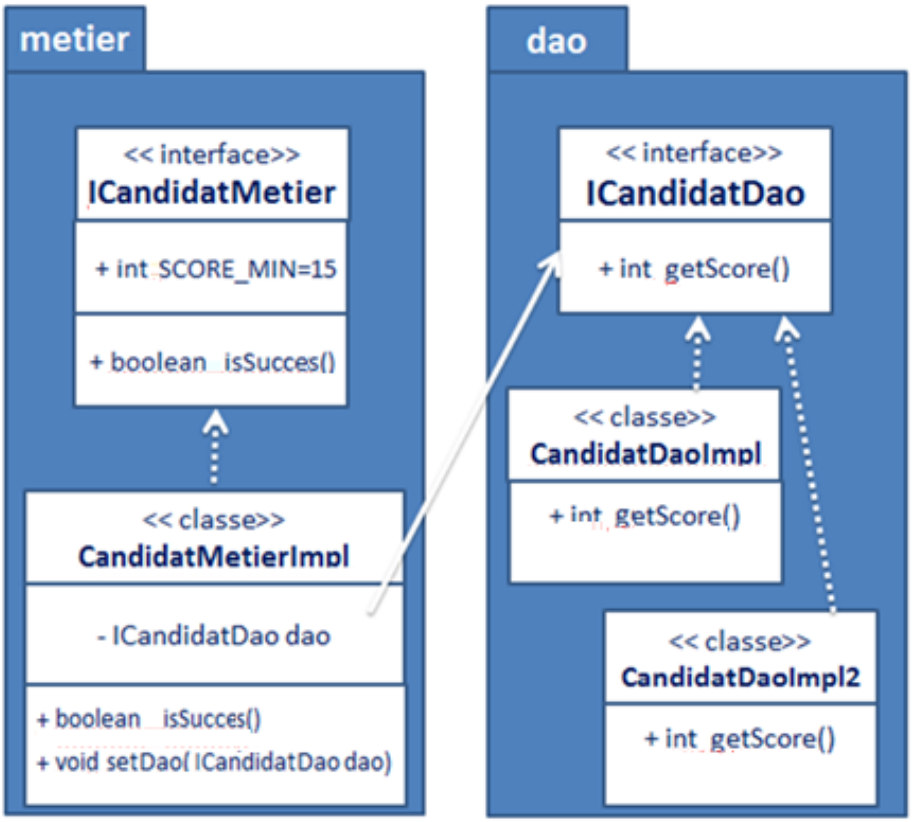
## Objectifs

Manipuler le concept d'injection de dépendances

- IoC par instanciation statique
- Ioc par instanciation dynamique par fichier texte
- IoC Avec Spring par configuration XML
- IoC avec Spring par Annotation
- Ioc avec Spring par Maven
- Ioc avec Spring Boot

## Modèle Conceptuel et notions théoriques

Dans le but de présenter le mécanisme d'injection de dépendance, nous admettons le model conceptuel suivant :



Une implémentation « **CandidatMetierImpl** » de l'interface « **ICandidatMetier** » présente une dépendance avec une instance « **dao** » de type « **ICandidatDao** » : La méthode « **isSucces ()** » appelle la méthode « **getScore()** » de l'objet « **dao** » pour retourner un booléen selon la valeur du score récupérée.

- Dans notre exemple, Il existe deux implémentations différentes pour l'interface « **ICandidatDao** » :

  ✓ « **CandidatDaoImpl** » : qui lit la valeur du score à partir d'une base de données (à ne pas coder cette procédure d'accès à la BD)
  ✓ « **CandidatDaoImpl2** » : qui lit la valeur du score à partir d'un service web (à ne pas coder cette procédure d'accès au Service Web).

- Puisque cette dépendance est liée à une interface, elle est dite **indirecte** (faible couplage) : La classe métier « **CandidatMetierImpl** » est dite, alors :
  ✓ **fermée à la modification** : son code source reste indépendant du changement et de la modification des implémentations de l'interface « **ICandidatDao** » .
  ✓ **ouverte à l'extension** : elle accepte de nouveaux types d'implémentation pour l'interface « **ICandidatDao** ».

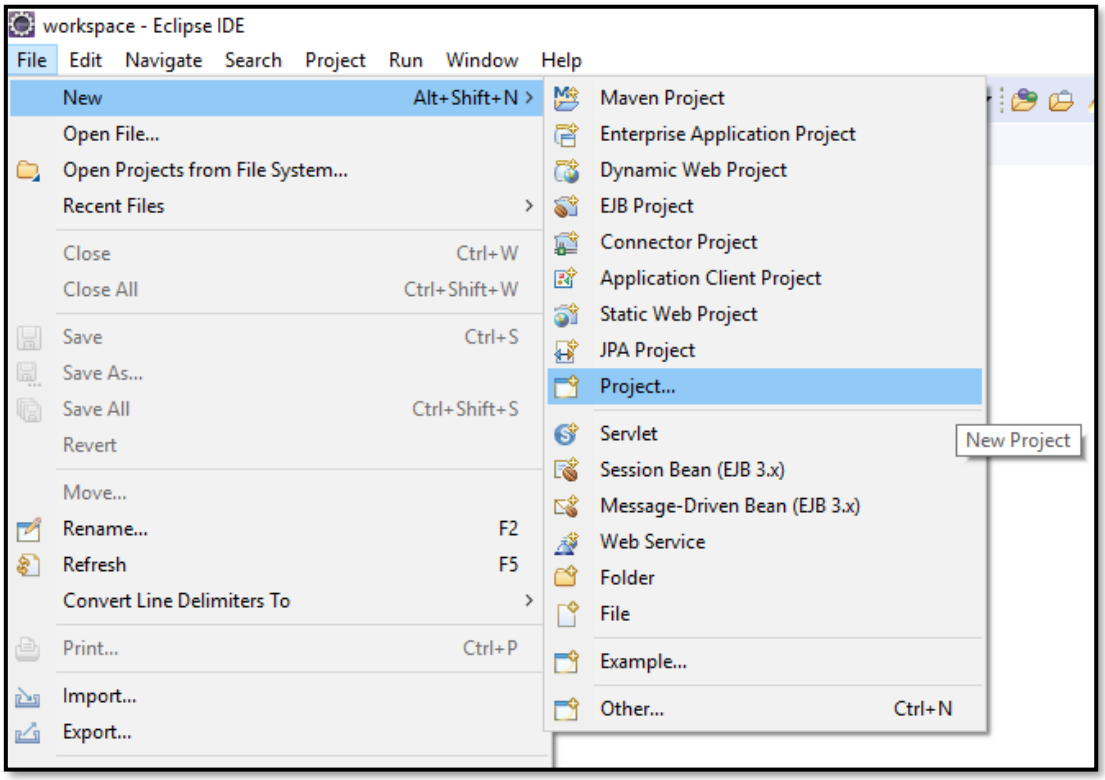
- La classe « **CandidatMetierImpl** » a besoin d'un objet de type « **ICandidatDao** » qui sera injecté via la méthode « **setDao(..)** ». On parle ainsi d'une **injection de dépendance** ( ou **IoC** : **inversion de contrôle**) **par méthode** (setter).

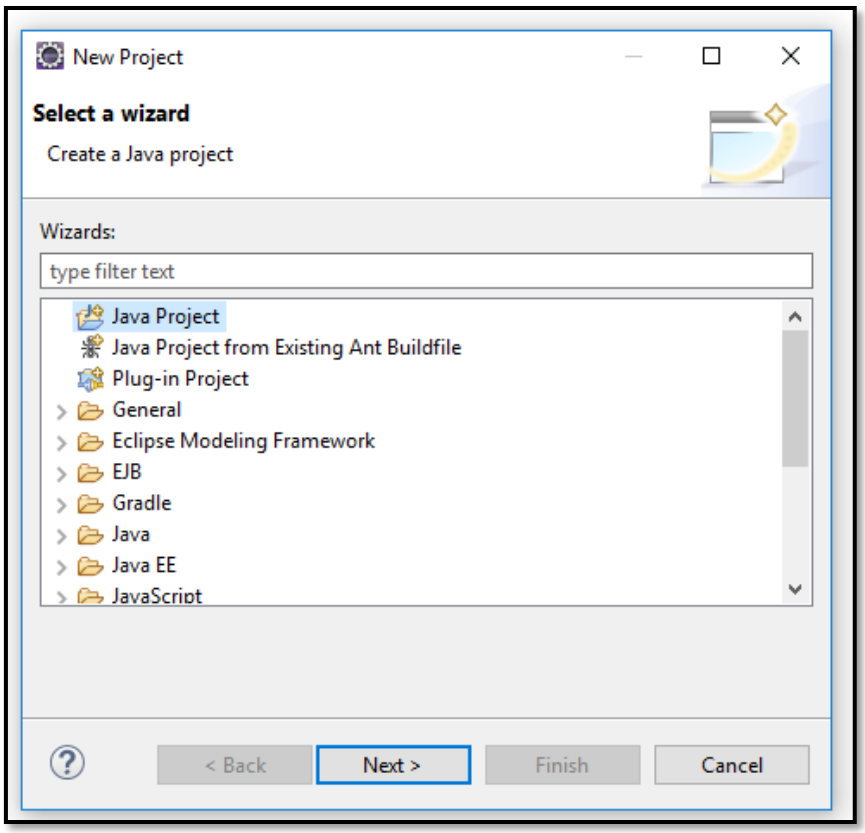
Dans cet atelier, on présente diverses méthodes pour réaliser cette injection :

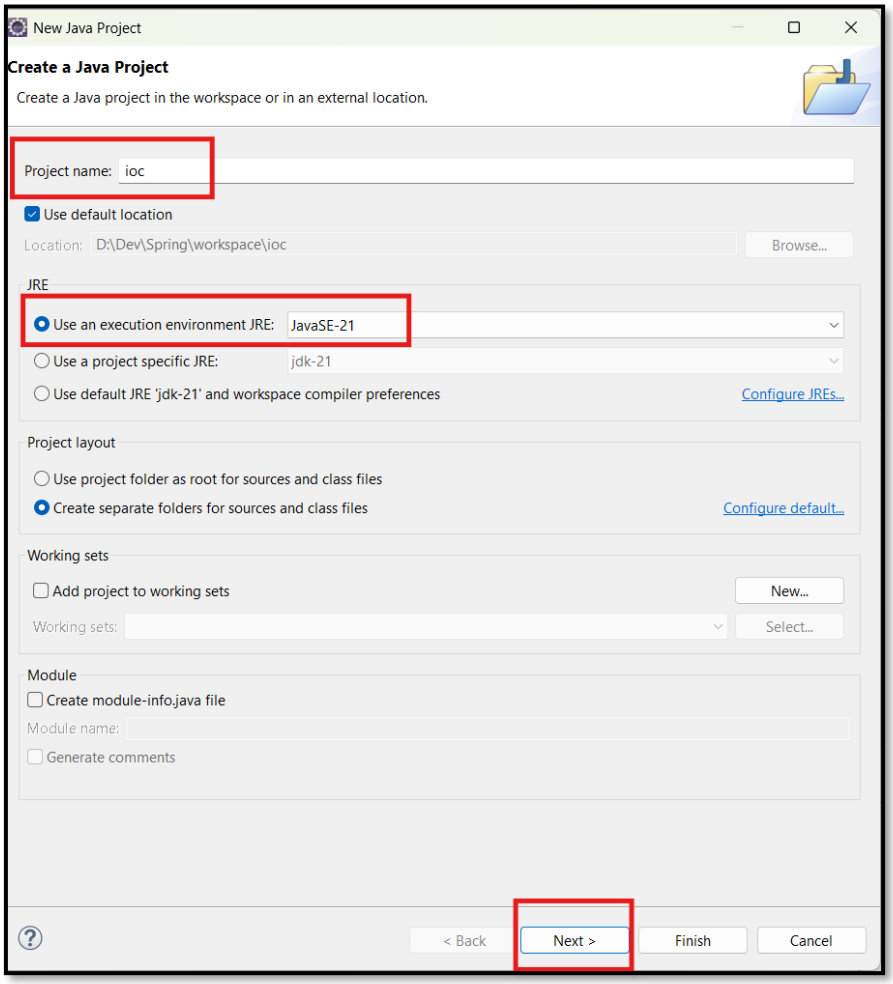
1. **Par instanciation statique**
2. **Par instanciation dynamique**
3. **Par l'outil Spring en utilisant une configuration XML**
4. **Par l'outil Spring en utilisant des annotation**
5. **Par l'outil Spring en utilisant des annotations par Maven**

# A. IoC par instanciation statique

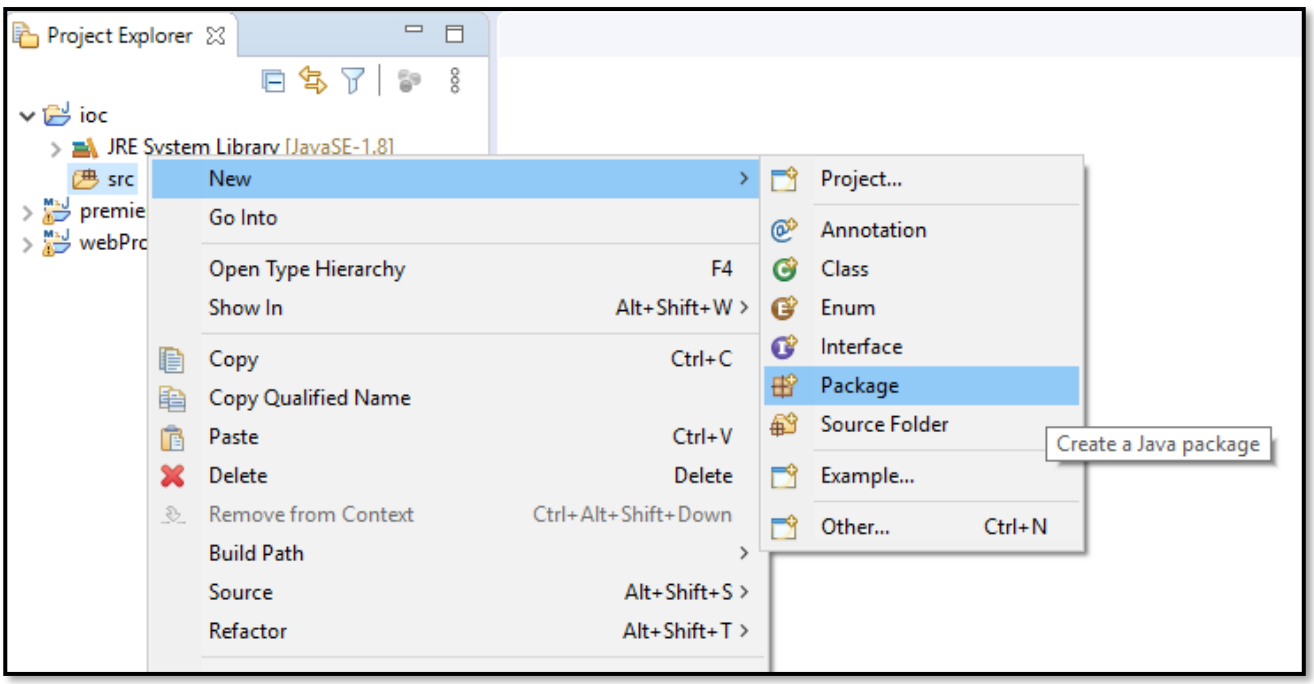
1. Créer un nouveau projet de type « **Java Project** » nommé «**ioc**» dans votre **workspace** :

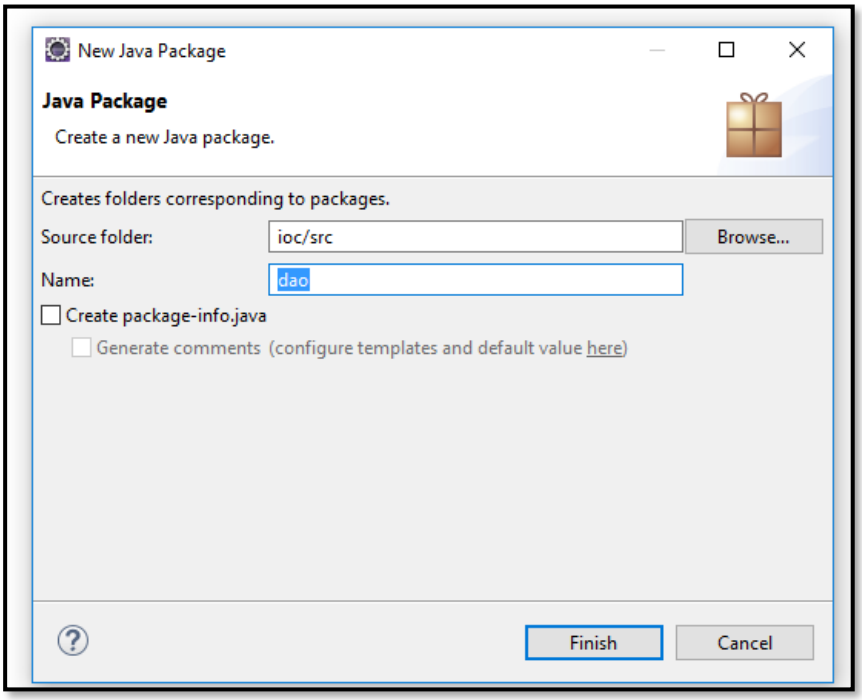





## 2. Créer un package «**dao**» :






3. Définir, dans le package «**dao**», les éléments suivants (une interface et ses deux implémentations)

```java
package dao;

public interface ICandidatDao
{
    int getScore();
}
```

```java
package dao;

public class CandidatDaoImpl implements ICandidatDao
{
    public int getScore()
    {
        /*
         * récupérer la valeur à partir d'une base de données
         */
        int s =12;  //exemple de valeur
        return s;
    }
}
```

```java
package dao;
public class CandidatDaoImpl2 implements ICandidatDao {
     public int getScore()
     {
          /*
           * lire la valeur à partir d'un service web
           */
          int s =18;  //exemple de valeur
          return s;
     }
}
```

4. Créer un autre package «**metier**» pour définir les éléments suivants :

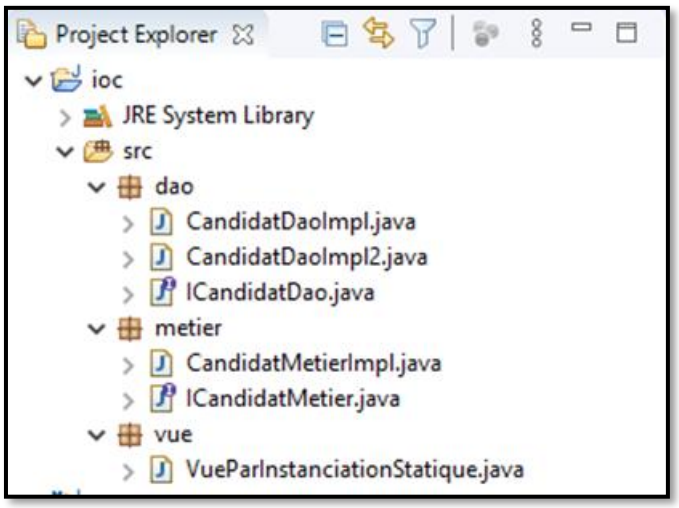
```java
package metier;
public interface ICandidatMetier
{
  // constante qui définit le score minimum pour la réussite
   final int SCORE_MIN =15;

   boolean isSucces();
}
```

```java
package metier;
import dao.ICandidatDao;
public class CandidatMetierImpl implements ICandidatMetier {
     ICandidatDao dao =null;
     //méthode pour l'injection de dépendance
     public void setDao(ICandidatDao dao)
     {
          this.dao = dao;
     }
     //Redéfinition de la méthode isSucces()
     public boolean isSucces()
     {
          int score= dao.getScore();
//retourner «true» si le score est supérieur ou égal au score minimum, sinon retourner «false»
          return (score>=SCORE_MIN);
     }
}
```

5. Créer un package « **vue** » pour réaliser l'injection de dépendance par instanciation statique à travers la classe suivante :

```
package vue;

import dao.CandidatDaoImpl;
import metier.CandidatMetierImpl;

public class VueParInstanciationStatique {

	public static void main(String[] args) {
		CandidatDaoImpl dao = new CandidatDaoImpl();
		CandidatMetierImpl metier = new CandidatMetierImpl();
		/*
		 * Injection de dépendance par instanciation statique
		 */
		metier.setDao(dao); //injection de dépendance par méthode (setter)
		boolean res= metier.isSucces();
		if (res)
			System.out.println("Candidat réussi");
		else
			System.out.println("Candidat non réussi");
	}
}
```

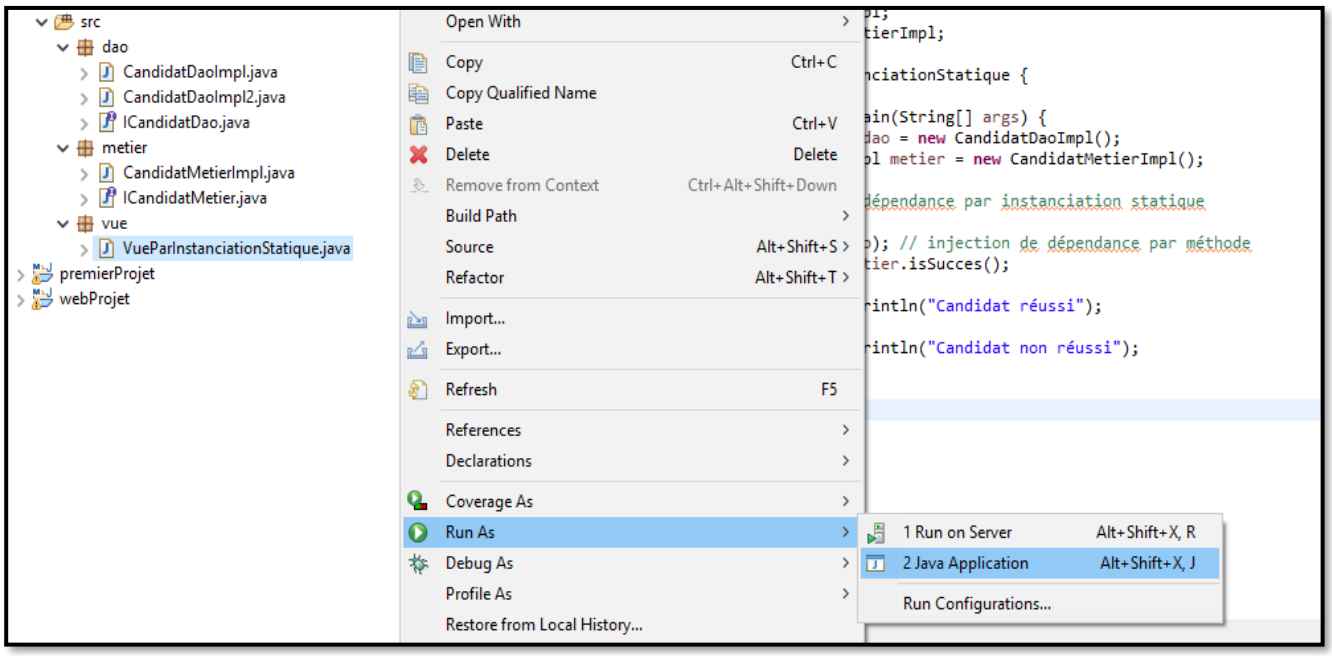
- Voici la vue globale du dossier « src » :



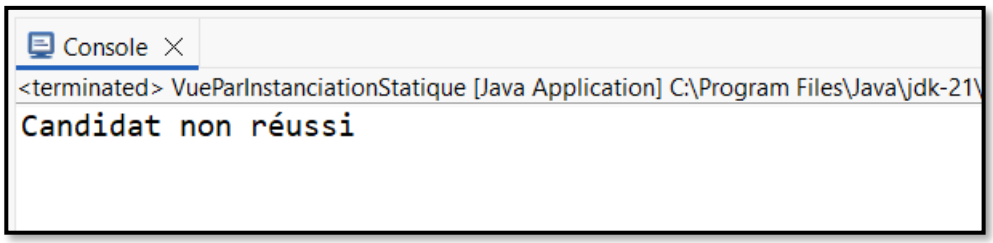

- Remarquer que la classe « **VueParInstanciationStatique** » comporte des **dépendances directes** vers les deux classes « **CandidatMetierImpl** » et « **CandidatDaoImpl**».

- Les deux objets « **dao** » et « **metier** » sont instanciés d'une manière statique (en utilisant l'opérateur **new**).

6. Lancer l'exécution de la vue :




Et visualiser le résultat :

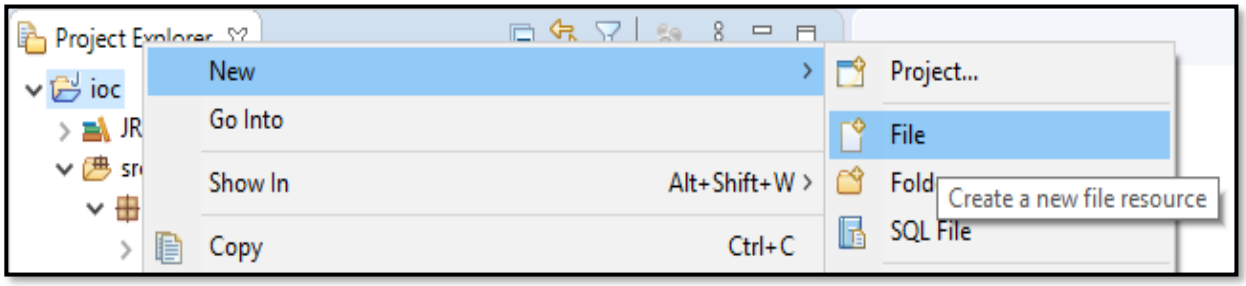



# B. Ioc par instanciation dynamique par fichier texte

Nous visons maintenant à réaliser le même traitement mais avec **instanciation dynamique** des objets « **dao** » et « **metier** » :

- les noms des classes d'implémentation à utiliser seront déclarés à l'extérieur du code source dans un **fichier** texte de configuration.
- Le programme récupère, au moment de son exécution, les noms des classes à instancier à partir d' fichier texte donné.

7. Sélectionner la racine du projet « **ioc** » et ajouter un fichier nommé « **config.txt** » à travers la commande « **New/File** » :




**8.** Dans ce fichier, placer les deux lignes suivantes :

```
dao.CandidatDaoImpl
metier.CandidatMetierImpl
```

- ✓ La première ligne indique le nom de la classe à instancier pour le « **dao** ».
- ✓ La deuxième ligne indique le nom de la classe à instancier pour le « **metier** ».

**9.** La classe suivante, dans le package « **vue** », constitue une nouvelle version qui permet de réaliser l'injection de dépendance par **instanciation dynamique** (au moment de l'exécution le programme détecte les noms des classes d'implémentation à utiliser) :

```
package vue;
import java.io.File;
import java.lang.reflect.Method;
import java.util.Scanner;
import dao.ICandidatDao;
import metier.ICandidatMetier;
public class VueParInstanciationDynamique {
    public static void main(String[] args) {
        /*
         * Injection de dépendance par instanciation dynamique (fichier texte)
         */
    try {
      Scanner scanner = new Scanner(new File("config.txt"));
      // lire le nom de la classe Dao
      String daoCandidatClassName= scanner.nextLine();
      //Charger la classe Dao
      Class classeDao= Class.forName(daoCandidatClassName);
      // Créer une instance de cette calsse
      ICandidatDao dao =(ICandidatDao) classeDao.newInstance();
      // lire le nom de la classe Metier
      String metierCandidatClassName= scanner.nextLine();
      //Charger la classe Metier
      Class classeMetier= Class.forName(metierCandidatClassName);
      // Créer une instance de cette calsse
      ICandidatMetier metier =(ICandidatMetier) classeMetier.newInstance();
      //Appel dynamiquement de la méthode setDao
      Method m = classeMetier.getMethod("setDao", ICandidatDao.class);
      m.invoke(metier, dao);
      boolean res= metier.isSucces();
```

```
        if (res)
            System.out.println("Candidat réussi");
        else
                System.out.println("Candidat non réussi");

            } catch (Exception e) {    e.printStackTrace();
            }
    }}
```

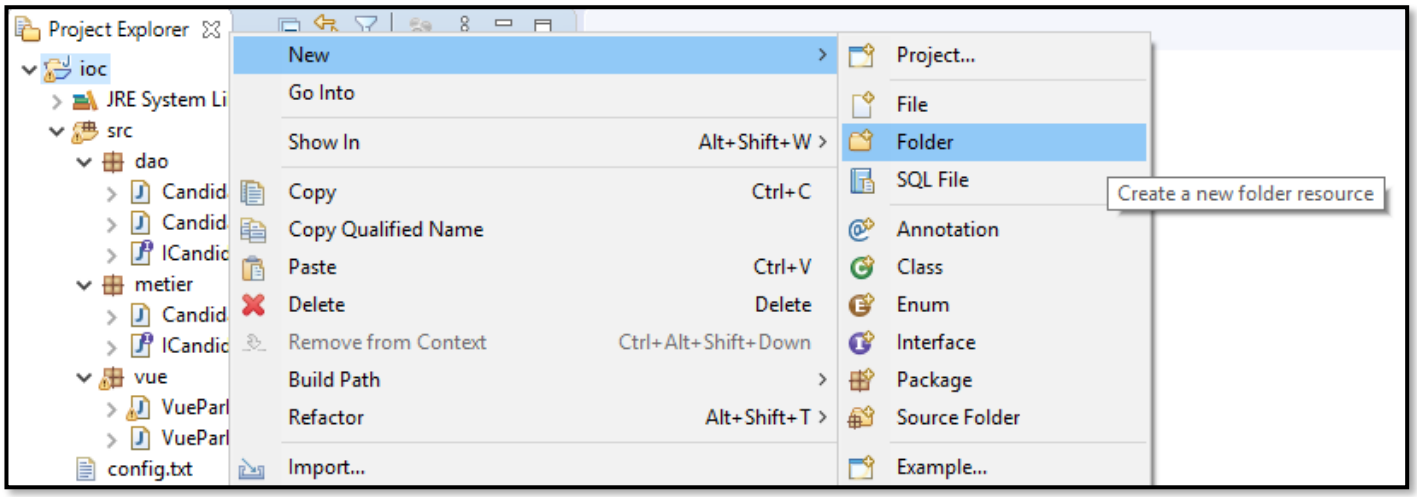
**10.** Exécuter la nouvelle vue et visualiser le résultat. Pour modifier l'implémentation de l'interface « **ICandidatDao** », il suffit d'accéder au fichier « **config.txt** » et changer le nom de la classe à utiliser (ex : « **CandidatDaoImpl2** » sans toucher le code source de la classe d'instanciation.

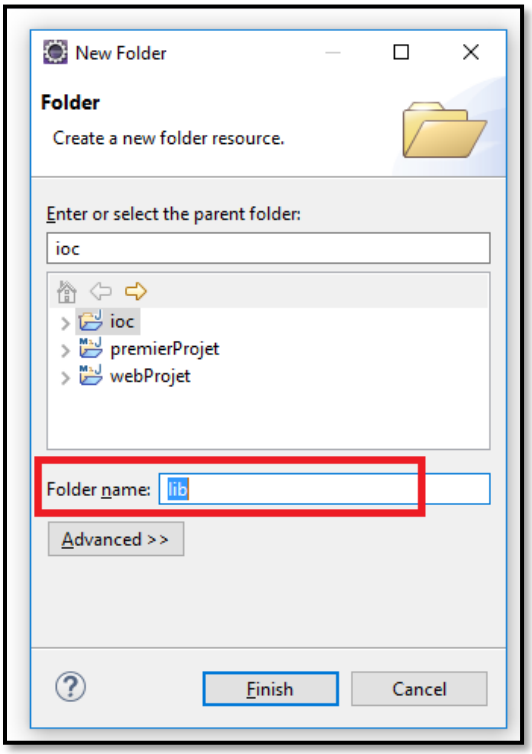
# C.IoC Avec Spring par configuration XML

Pour développer avec "**Spring**» **par XML**, ajouter les bibliothèques nécessaires. Pour se faire :

**11.** sélectionner, dans "**eclipse**", la racine du projet "**ioc**"

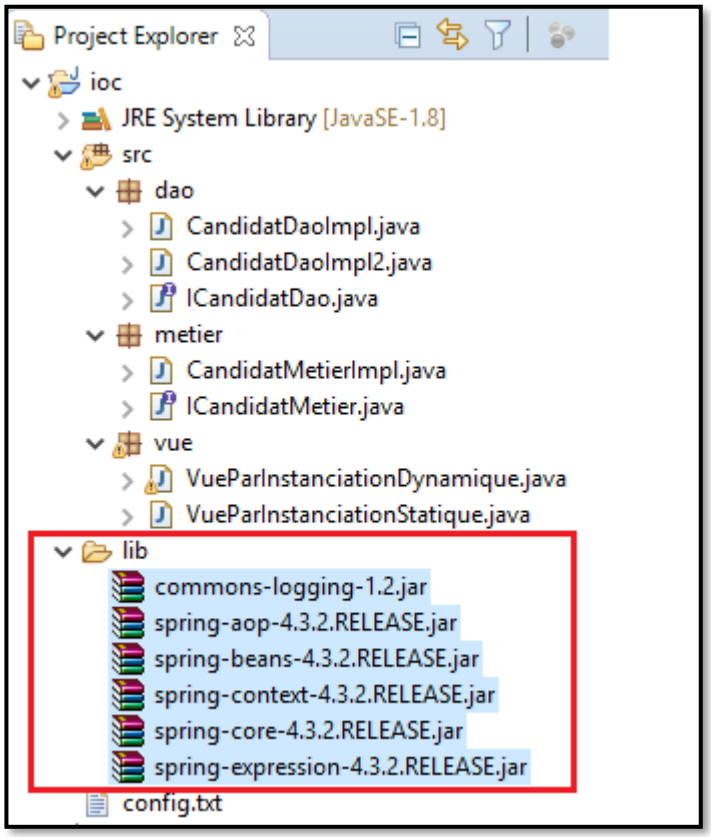
**12.** Ajouter un nouveau dossier (**New / folder**) nommé "**lib**"



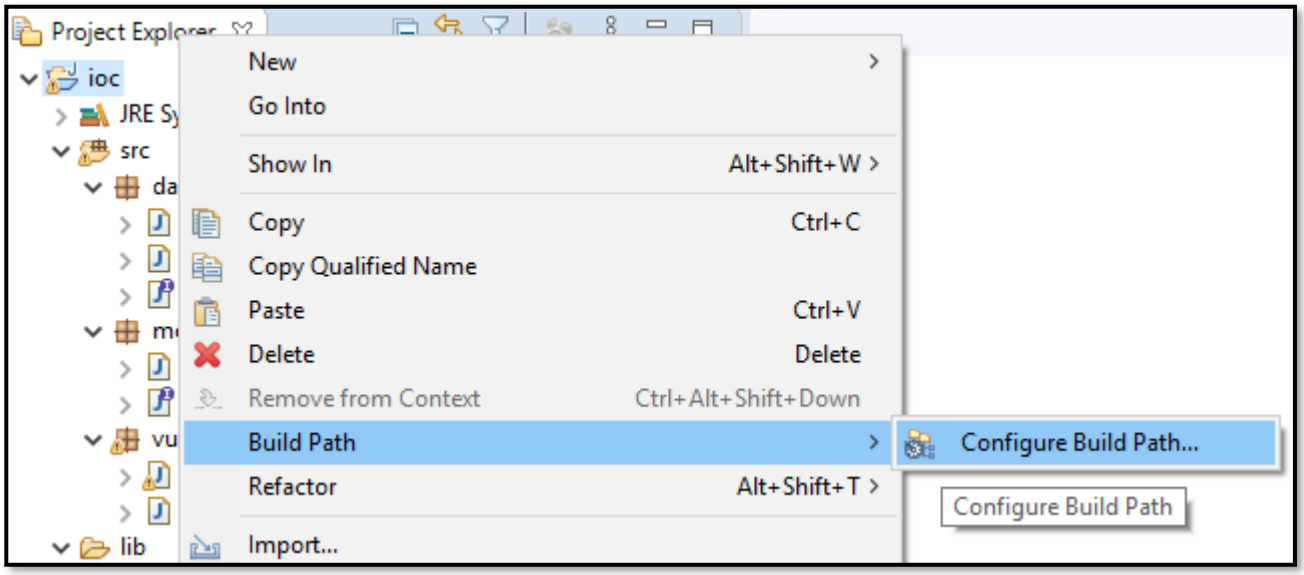



**13.** Ajouter dans ce dossier « **lib** » les fichiers **.jar** distribués avec cet atelier dans le dossier « **lib** » compressé dans un fichier « **pieces_jointes.zip** » (Configuration manuelle)**.**

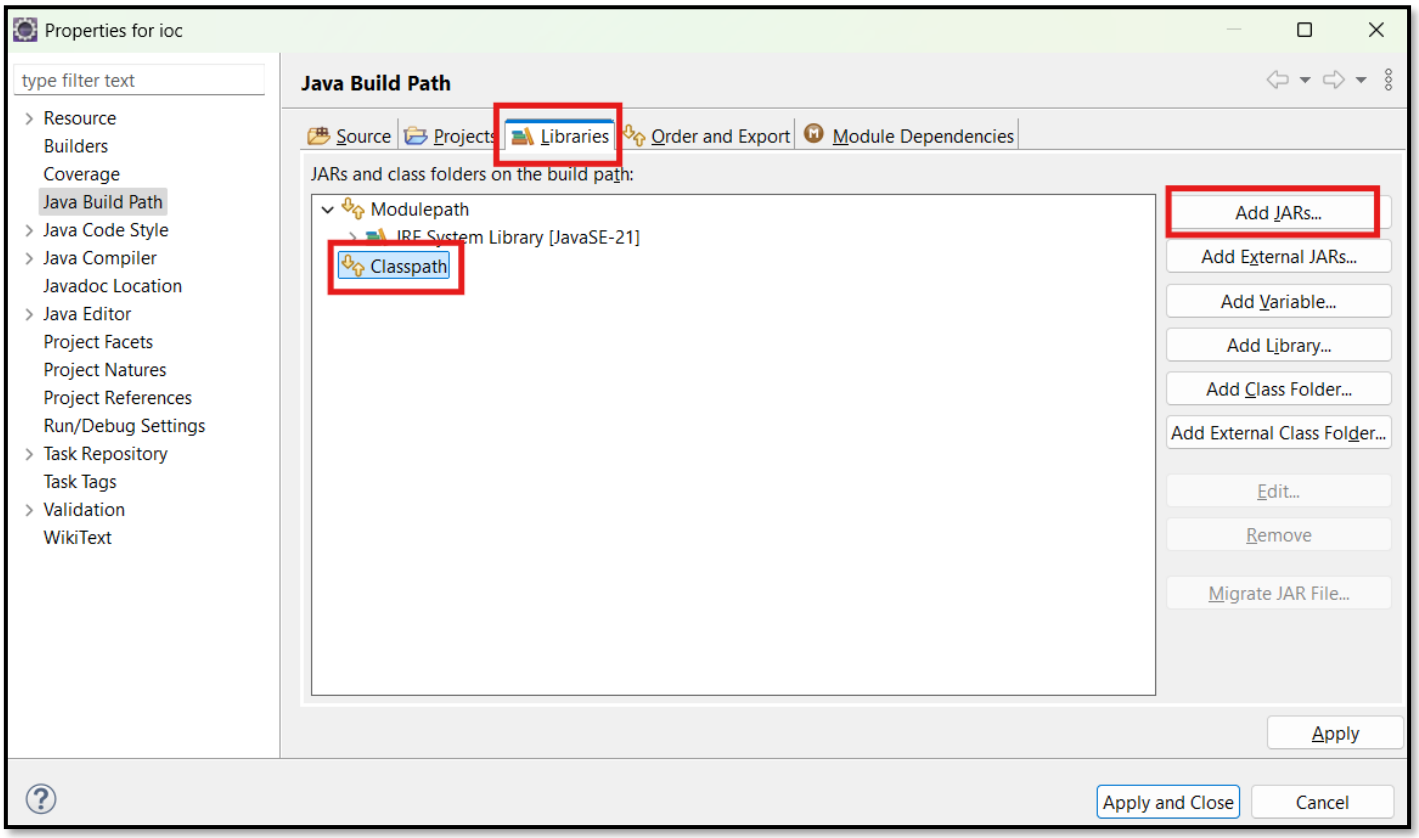
**14.** Actualiser le projet «**ioc**» pour visualiser la liste des bibliothèques dans le dossier «**lib**»



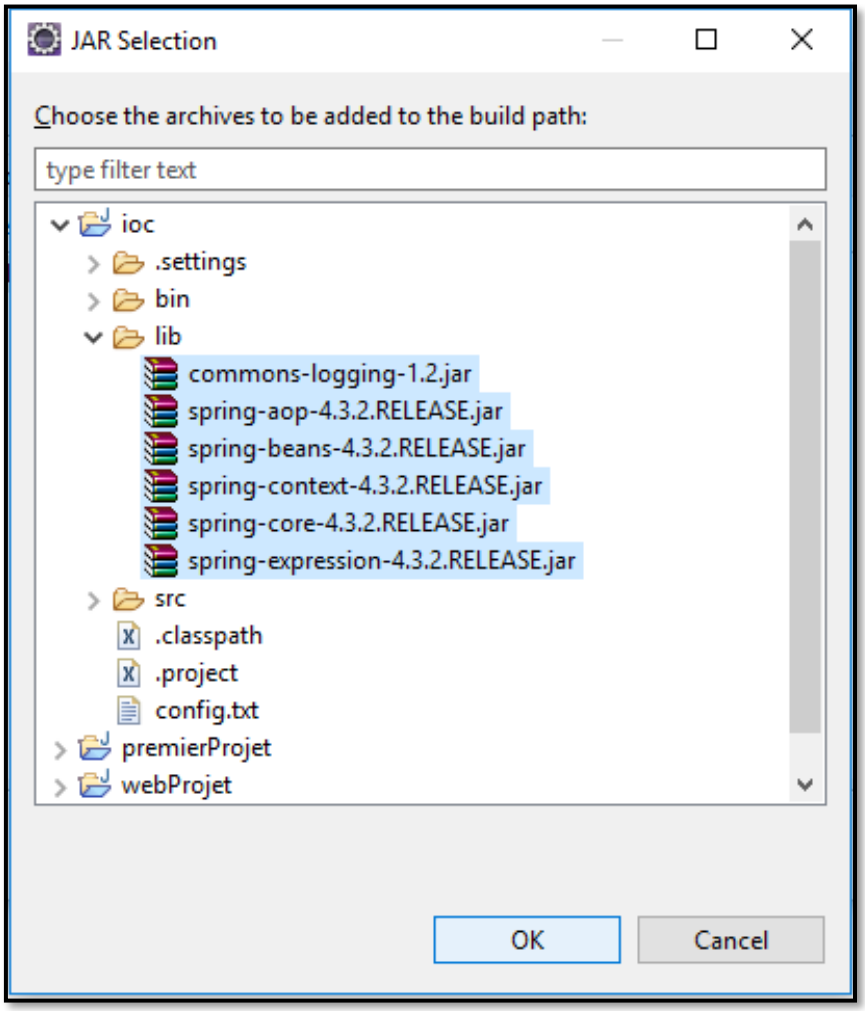
**15.**Sélectionner la racine du projet, puis avec le bouton droit, choisir la commande "**BuildPath/ Configure Build Path**" :

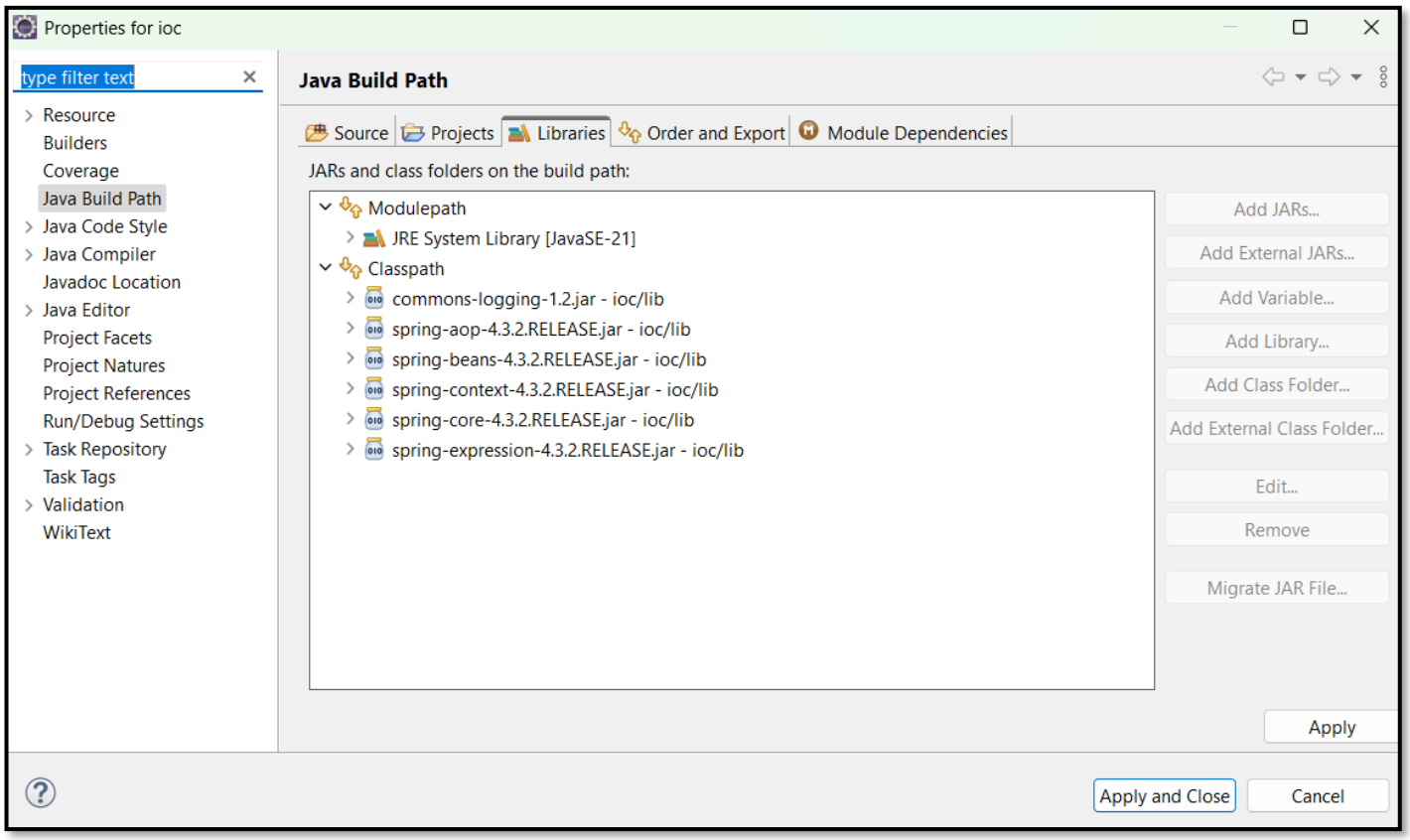



**16.**Aller à l'onglet "**Libraries**", sélectionner "**Classpath**" et choisr "**Add Jars..**" :



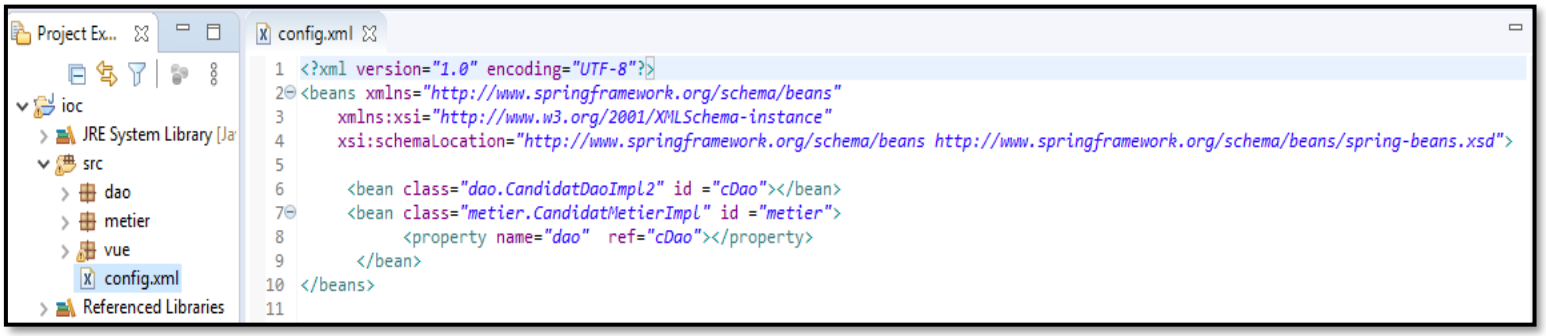

**17.**Sélectionner les fichiers "**jars**" du dossier "**lib**" dernièrement créé:


JAR Selection
Choose the archives to be added to the build path:
type filter text
ioc
.settings
bin
lib
commons-logging-1.2.jar
spring-aop-4.3.2.RELEASE.jar
spring-beans-4.3.2.RELEASE.jar
spring-context-4.3.2.RELEASE.jar
spring-core-4.3.2.RELEASE.jar
spring-expression-4.3.2.RELEASE.jar
src
.classpath
.project
config.txt
premierProjet
webProjet
OK
Cancel


Properties for ioc
type filter text
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Project Facets
Project Natures
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
Module Dependencies
JARs and class folders on the build path:
Modulepath
JRE System Library [JavaSE-21]
Classpath
commons-logging-1.2.jar - ioc/lib
spring-aop-4.3.2.RELEASE.jar - ioc/lib
spring-beans-4.3.2.RELEASE.jar - ioc/lib
spring-context-4.3.2.RELEASE.jar - ioc/lib
spring-core-4.3.2.RELEASE.jar - ioc/lib
spring-expression-4.3.2.RELEASE.jar - ioc/lib
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel

**18.**Sélectionner le dossier "**src**" du projet "**ioc**".

**19.**Copier/coller le fichier « **config.xml** », donné avec l'énoncé, dans le dossier "**src**" du projet "**ioc**" (sous eclipse) :




**20.**Dans le fichier « **config.xml** », remarquer les instructions suivantes (à l'intérieur de la balise <**beans**>:

```
    <bean class="dao.CandidatDaoImpl2" id ="cDao"></bean>
    <bean class="metier.CandidatMetierImpl" id ="metier">
          <property name="dao"  ref="cDao"></property>
   </bean>
```

Il s'agit de la déclaration de deux objets (**beans**):

- "**cDao**" de type "**dao.CandidatDaoImpl**"
- "**metier**" de type "**metier.CandidatMetierImpl**"

**L'injection de dépendance** est réalisée par l'intermédiaire de la propriété "**dao**".

**21.**Dans le packqge "**vue**", ajouter la classe "**VueParSpringXML**" ayant le code suivant:

```
package vue;

import org.springframework.context.*;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import metier.ICandidatMetier;

public class VueParSpringXML {
    public static void main(String[] args) {
    /*
     * Injection de dépendance avec Spring (Configuration XML)
     */
// Lire le fichier de configuration
```

```
        ApplicationContext springContext = new
ClassPathXmlApplicationContext("config.xml");

//récupérer un bean  de type "ICandidatMetier" en spécifiant son id "metier"
        ICandidatMetier metier =(ICandidatMetier)
springContext.getBean("metier");
        //Appeler la méthode isSucces()
        boolean res= metier.isSucces();
        if (res)
             System.out.println("Candidat réussi");
        else
             System.out.println("Candidat non réussi");
    }
}
```

**22.** Lancer l'exécution de cette vue et interpréter le résultat.

**23.** Choisir une autre implémentation de l'interface "**ICandidatDao**" et tester.

# D. Ioc avec Spring par Annotation

Maintenant passons à une autre méthode utilisée par « **Spring** » pour réaliser l'Ioc (Inversion de contrôle ou injection de dépendance); il s'agit de l'utilisation **des annotations.** Pour se faire:

**24.** Ajouter, dans le package "**dao**", une nouvelle implémentation de l'interface "**ICandidatDao**":

```
package dao;

import org.springframework.stereotype.Repository;

@Repository
public class CandidatDaoAnnotationImpl implements ICandidatDao
{

  public int getScore() {
       /*
        * lire la valeur à partir d'un fichier texte
        */
       return 20;
  }
}
```

- L'annotation « **@Repository** » permet d'informer « **Spring** » que la classe est un **Spring BEAN.**
- L'annotation « **@Repository** » permet de marquer la classe pour être utilisée par la suite par « **Spring** » en cas de besoin d'une instance d'une classe qui implémente l'interface « **ICandidatDao** ».

**25.** Ajouter, dans le package "**metier**", une nouvelle implémentation de l'interface "**ICandidatMetier**":

```
package metier;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import dao.ICandidatDao;

@Service
public class CandidatMetierAnnotationImpl implements
ICandidatMetier {

    @Autowired //pour injecter un bean  de type ICandidatDao
    ICandidatDao dao =null;

    public void setDao(ICandidatDao dao) {
        this.dao = dao;
    }

    public boolean isSucces() {
         int score= dao.getScore();
        return (score>=SCORE_MIN);
    }
}
```

- L'annotation « **@Service** » permet d'informer « **Spring** » que la classe est un **Spring BEAN.**

- L'annotation « **@Service** » marque la classe pour être utilisée par la suite par « **Spring** » en cas de besoin d'une instance d'une classe qui implémente l'interface « **ICandidatMetier** ».
- **NB** : **@Repository**, **@Service** et **@Component** présentent pratiquement la même fonction de déclaration des beans (objets) pour « **Spring** ».
- Les « **Spring BEAN »** créés sont générés dans « **Spring IoC Container »** (Le conteneur Spring IoC).
- L'annotation « **@Autowired** » indique l'emplacement de l'injection de dépendance.
- « **Spring** » cherche une classe déclarée (avec l'une des annotations **@Repository** ou **@Service** ou **@Component**) qui implémente l'interface de déclaration de l'attribut ayant l'annotation « **@Autowired** ».
- Dans ce cas, c'est l'attribut « **dao** » qui est annoté par « **@Autowired** », Spring va instancier d'une manière dynamique la classe « **CandidatDaoAnnotationImpl** » (ayant l'annotation « **@Repository** » et implémentant l'interface « **ICandidatDao** ») puis affecter cette instance à l'attribut « **dao** » de type « **ICandidatDao** ».
- En cas de l'existence de plusieurs classes annotées par « **@Repository** » et qui implémentent l'interface « **ICandidatDao** », on procède comme suit :
  - ✓ Modifier la classe «**CandidatMetierAnnotationImpl**» par l'ajout d' une autre annotation « **@Resource** » juste avant la déclaration de l'attribut « **dao** » pour indiquer le nom du bean (objet) à utiliser (par exemple « **daoBean** » :

```
package metier;

import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import dao.ICandidatDao;
```

```java
@Service
public class CandidatMetierAnnotationImpl implements
ICandidatMetier
{

     @Autowired //pour injecter un bean de type ICandidatDao
     ICandidatDao dao =null;

     public void setDao(ICandidatDao dao)
{
          this.dao = dao;
     }

     public boolean isSucces() {
           int score= dao.getScore();
          return (score>=SCORE_MIN);
     }
}
```

- ✓ L'annotation « @Resource » nécessite l'ajout d'une bibliothèque au Classpath du projet : **javax.annotation-api-1.3.2.jar** (donné en pièce jointe avec l'atelier)
- ✓ Ceci nécessite l'existence d'un bean ayant le nom « **daoBean** » : Au moment de la déclaration d'un bean, spécifier son nom comme suit :

```java
package dao;

import org.springframework.stereotype.Repository;

@Repository (value="daoBean")
public class CandidatDaoAnnotationImpl implements ICandidatDao {

     public int getScore() {
          /*
           * lire la valeur à partir d'un fichier texte
           */
          return 20;
     }
}
```

**26.** Dans le package "**vue**", ajouter la classe " **VueParSpringAnnotation**" ayant le code suivant puis lancer l'exécution :

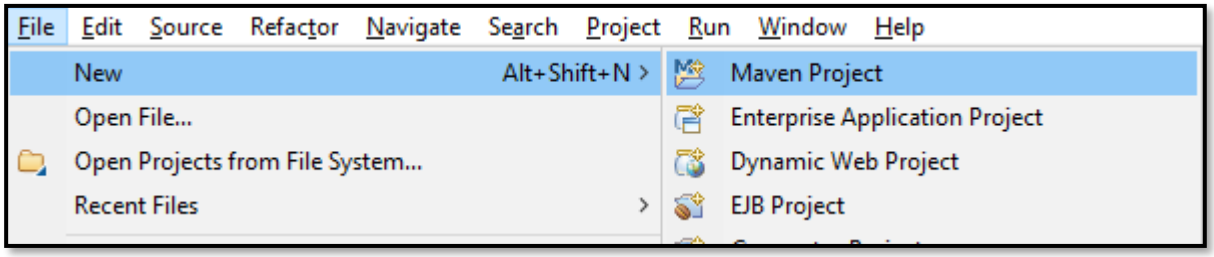
```
package vue;
import org.springframework.context.*;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import metier.ICandidatMetier;

public class VueParSpringAnnotation {

     public static void main(String[] args) {
          /*
          * Injection de dépendance avec Spring par Annotation
          */
// Spécifier les packages dans lesquels Spring cherche et lit les annotations
          ApplicationContext springContext = new
AnnotationConfigApplicationContext("dao", "metier");
          //récupérer un bean en spécifiant uniquement son interface
          ICandidatMetier metier =(ICandidatMetier)
springContext.getBean(ICandidatMetier.class);
          //Appeler la méthode isSucces()
          boolean res= metier.isSucces();
          if (res)
               System.out.println("Candidat réussi");
          else
               System.out.println("Candidat non réussi");
     }}
```

**27.** Lancer l'exécution de cette vue et interpréter le résultat.

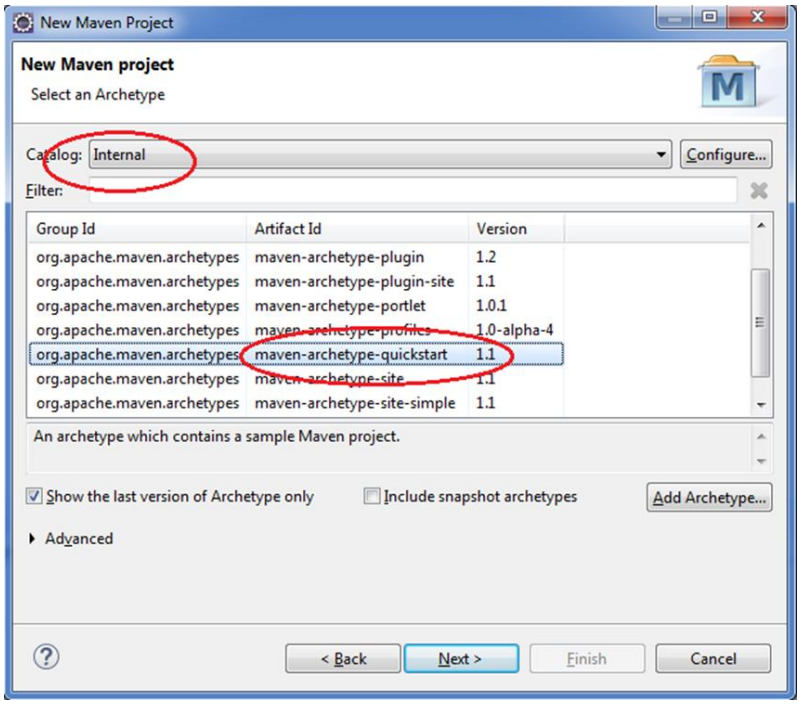
**28.** Proposer une autre implémentation de l'interface "**ICandidatDao**» avec l'utilisation de l'annotation « **@Repository** » et tester.

# E. Ioc avec Spring par Maven

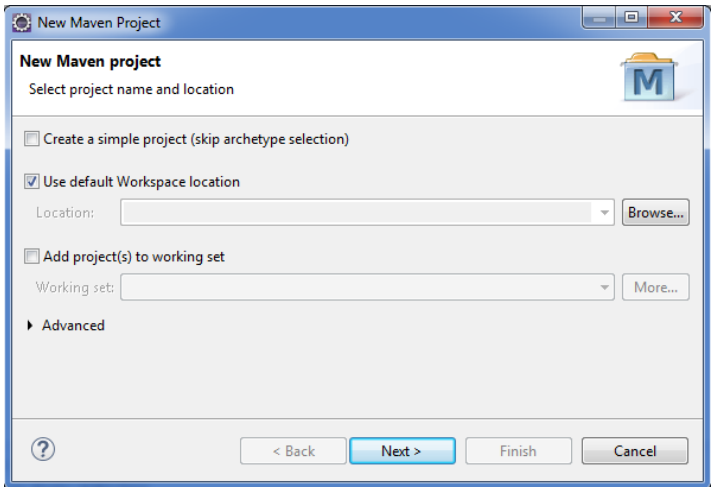
**29.** Créer un nouveau projet **Maven** nommé « **ioc-maven-spring** » :

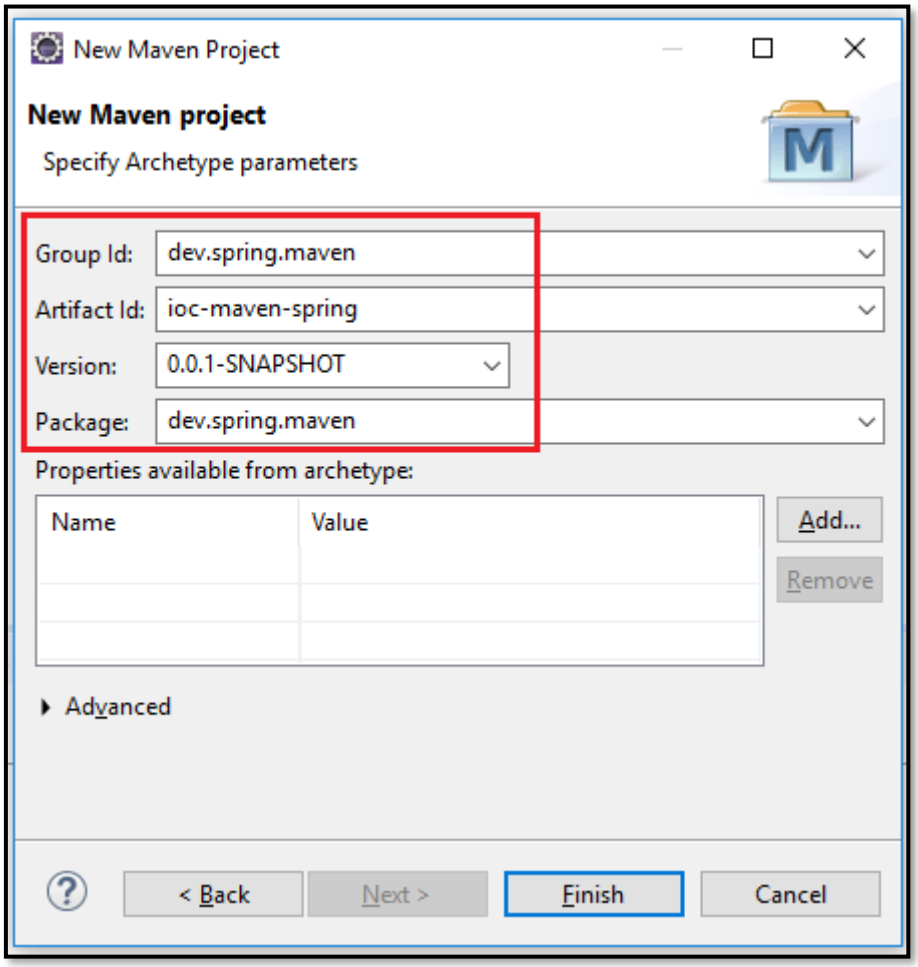



**30.** Appuyer sur « **Next** » et choisir le modèle « **maven-archetype-quickstart** » :



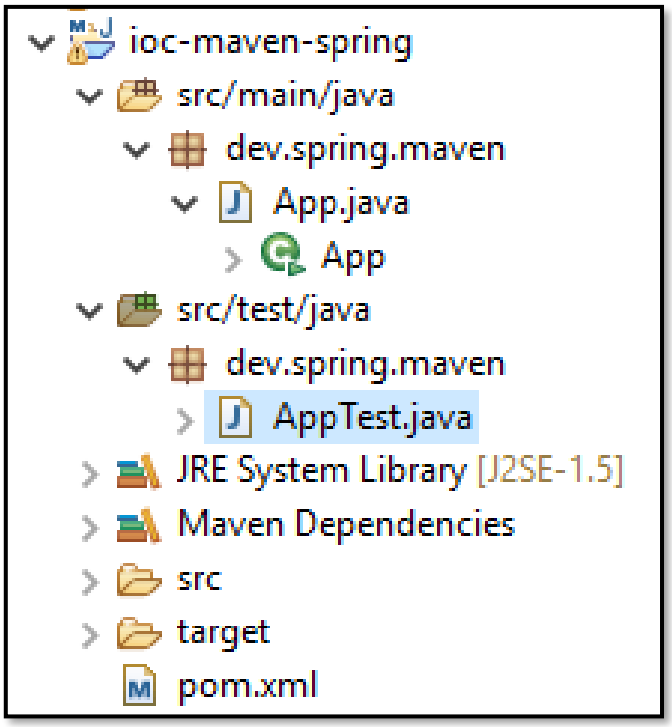

**31.** Spécifier les propriétés du projet comme suit :






**32.**Appuyer sur « **Finish** » pour achever la création du projet.

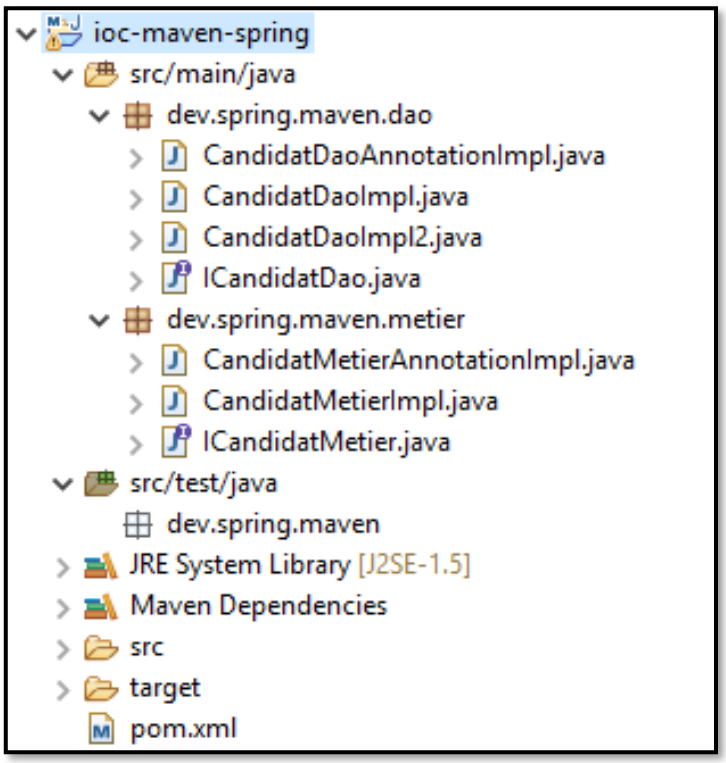
**33.**Voici la structure du projet :




**34.** Editer le fichier « **pom.xml** » pour ajouter les deux dépendances suivantes :

```
<!-- Spring Core -->
<!-- http://mvnrepository.com/artifact/org.springframework/spring-core -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>4.1.4.RELEASE</version>
        </dependency>

<!-- Spring Context -->
<!-- http://mvnrepository.com/artifact/org.springframework/spring-context -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.1.4.RELEASE</version>
        </dependency>
```

35. Supprimer les deux fichiers « **App.java** » et « **AppTest.java** » (inutiles).
36. Créer un package « **dev.spring.maven.dao**» sous « **src/main/java** ».
37. Copier le contenu du package « **dao** » du projet « **ioc** » et le coller dans le package « **dev.spring.maven.dao** » du projet « **ioc-maven-spring** ».
38. Créer un package « **dev.spring.maven.metier**» sous « **src/main/java** ».
39. Copier le contenu du package « **metier** » du projet « **ioc** » et le coller dans le package « **dev.spring.maven.metier** » du projet « **ioc-maven-spring** ». (Réaliser les modifications nécessaires au niveau des noms de packages pour les instructions « **import** ») :




40. Créer un package « **dev.spring.maven.vue**» sous « **src/main/java** ».

41. Créer une classe «**VueParSpringAnnotation**» dans le package « **dev.spring.maven.vue**» :

```
package dev.spring.maven.vue;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import dev.spring.maven.metier.ICandidatMetier;

public class VueParSpringAnnotationContexte {

	public static void main(String[] args) {
		// Lire les annotations
		ApplicationContext springContext =
				new AnnotationConfigApplicationContext(
				 "dev.spring.maven.dao",
				"dev.spring.maven.metier");
		// récupérer un bean en spécifiant son interface
		ICandidatMetier metier = (ICandidatMetier)
springContext.getBean(ICandidatMetier.class);
		// Appeler la méthode isSucces()
		boolean res = metier.isSucces();
		if (res)
			System.out.println("Candidat réussi");
		else
			System.out.println("Candidat non réussi");
	}
}
```

42. Lancer l'exécution.
43. Créer un package « **dev.spring.maven.conf**» sous « **src/main/java** ».
44. Créer une classe «**AppConfiguration**» dans le package « **dev.spring.maven.conf**». Cette classe déclare les packages de lecture des annotations pour l'injection de dépendances :

```
package dev.spring.maven.conf;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan({"dev.spring.maven.dao", "dev.spring.maven.metier"})
public class AppConfiguration { }
```

45. Créer une classe «**VueParSpringAnnotationAvecConfiguration**» dans le package « **dev.spring.maven.vue**» :

```java
package dev.spring.maven.vue;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import dev.spring.maven.conf.AppConfiguration;
import dev.spring.maven.metier.ICandidatMetier;

public class VueParSpringAnnotationAvecConfiguration {
	public static void main(String[] args) {

		// créer un contexte à partir de la configuration contenue dans la classe
		// "AppConfiguration"
		ApplicationContext springContext = new
AnnotationConfigApplicationContext(AppConfiguration.class);
		// récupérer un bean en spécifiant son interface
		ICandidatMetier metier = (ICandidatMetier)
springContext.getBean(ICandidatMetier.class);
		// Appeler la méthode isSucces()
		boolean res = metier.isSucces();
		if (res)
			System.out.println("Candidat réussi...");
		else
			System.out.println("Candidat non réussi");
	}
}
```

**46.** Lancer l'exécution.

# F. Ioc avec Spring Boot

**47.** Prendre une copie du projet « **ioc-maven-spring** » et la nommer «**ioc-maven-spring-boot** ».

**48.** Remplacer le contenu du fichier « **pom.xml** » par le suivant et remarquer la déclaration des dépendances « **spring-boot-starter** » et « **spring-boot-starter-test** »:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/201/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.5.5</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>dev.ms.maven</groupId>
  <artifactId>ioc-maven-spring-boot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```
  <name>ioc-maven-spring-boot</name>
  <url>http://maven.apache.org</url>
     <properties>
          <java.version>17</java.version>
     </properties>
     <dependencies>
          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter</artifactId>
          </dependency>
          <dependency>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-starter-test</artifactId>
               <scope>test</scope>
          </dependency>

     </dependencies>

     <build>
          <plugins>
               <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
               </plugin>
          </plugins>
     </build>

</project>
```

**49.** Une erreur est détectée au niveau de la classe « **CandidatMetierAnnotationImpl** » : The import javax.annotation.Resource cannot be resolved. Pour résoudre ce problème, il est nécessaire d'ajouter, dans le fichier POM, une dépendance de **javax** (qui est maintenant distribuée hors JDK pour les versions récentes de JDK) :

```
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.2</version>
</dependency>
```

**50.** Ajouter, maintenant, une nouvelle vue « **VueParSpringBootApplication** » dans le package : « **dev.spring.maven.vue** » :

```java
package dev.spring.maven.vue;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.ComponentScan;

import dev.spring.maven.metier.ICandidatMetier;

@SpringBootApplication
public class VueParSpringBootApplication
{
	public static void main(String[] args) {

		ApplicationContext springContext =
SpringApplication.run(VueParSpringBootApplication.class, args);
		// récupérer un bean en spécifiant son interface
		ICandidatMetier metier = (ICandidatMetier)
springContext.getBean(ICandidatMetier.class);
		// Appeler la méthode isSucces()
		boolean res = metier.isSucces();
		if (res)
			System.out.println("Candidat réussi...");
		else
			System.out.println("Candidat non réussi");
	}
}
```

**51.** Lancer l'exécution et remarquer le déclenchement de l'exception suivante :

```
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException:
No qualifying bean of type 'dev.spring.maven.metier.ICandidatMetier' available
	at
org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBea
nFactory.java:394)
	at
org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBea
nFactory.java:385)
	at
org.springframework.context.support.AbstractApplicationContext.getBean(AbstractApplicationConte
xt.java:1290)
	at
dev.spring.maven.vue.VueParSpringBootApplication.main(VueParSpringBootApplication.java:17)
```

Ceci est dû au fait que Spring Boot n'arrive pas à trouver le bean à injecter. La classe principale «**VueParSpringBootApplication**» est définie dans le package «**dev.spring.maven.vue**». Donc, par défaut, Spring Boot va chercher les beans dans ce package et ses sous-packages.

Ce qui nécessite d'indiquer explicitement les packages où sont déclarés les beans. Pour se faire, ajouter l'annotation suivante juste après l'annotation « **@SpringBootApplication** » dans le code de la classe principale :

```
@ComponentScan(basePackages = "dev.spring.maven")
```

Ceci permet à Spring Boot d'aller chercher les beans à injecter dans la hiérarchie du package « **dev.spring.maven** » :

**52.** Relancer l'exécution et visualiser le résultat :

```
.   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.5)

INFO 128324 --- [           main] d.s.m.vue.VueParSpringBootApplication    : Starting VueParSpringBootApplication using
Java 21.0.8 with PID 128324 (D:\Dev\Spring\workspace\ioc-maven-spring-boot\target\classes started by HP in
D:\Dev\Spring\workspace\ioc-maven-spring-boot)
INFO 128324 --- [           main] d.s.m.vue.VueParSpringBootApplication    : No active profile set, falling back to 1
default profile: "default"
INFO 128324 --- [           main] d.s.m.vue.VueParSpringBootApplication    : Started VueParSpringBootApplication in
0.931 seconds (process running for 1.24)
Candidat réussi...
```

## Exercice

**Donner la commande Maven qui permet de lancer l'exécution de l'application**