

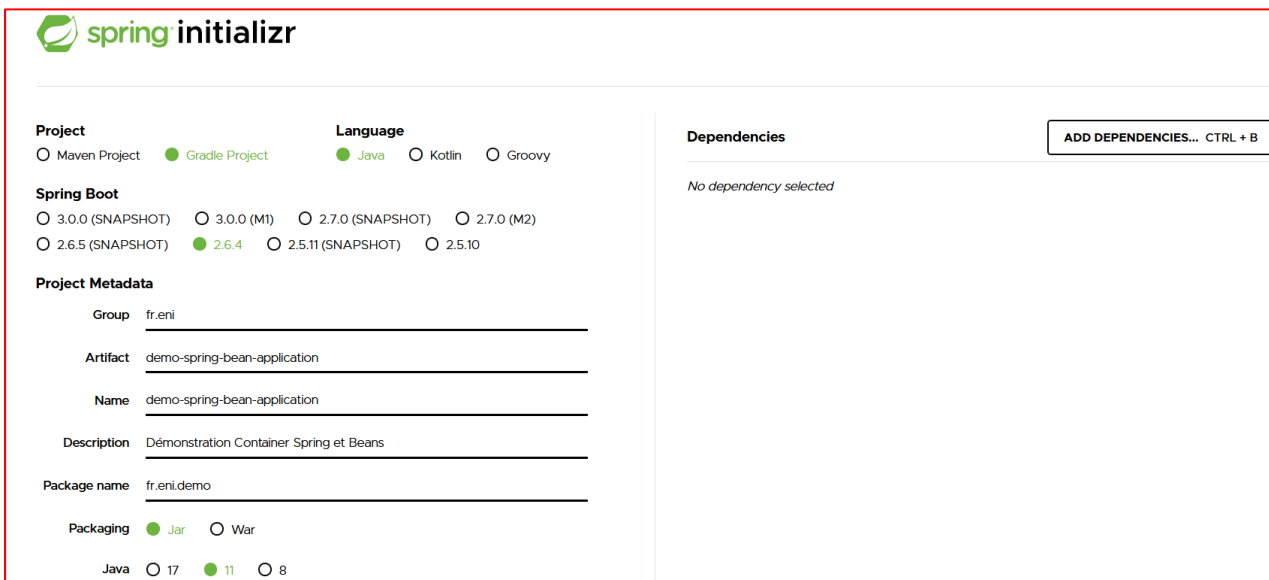
Conteneur Spring et Bean

Démonstration 2 du module 3

L'objectif de cette démonstration est de comprendre la notion de container Spring et de bean Spring.

Déroulement

Créer une application DemoSpringBeanApplication avec SpringBoot



The screenshot shows the Spring Initializr web application interface. It includes sections for Project (Maven or Gradle), Language (Java, Kotlin, or Groovy), Spring Boot version (3.0.0, 2.7.0, or 2.6.4), Project Metadata (Group, Artifact, Name, Description, Package name), and Packaging (Jar or War). The Dependencies section is currently empty, showing 'No dependency selected'.

Notion de conteneur Spring

- On peut récupérer une référence vers le conteneur Spring depuis la classe d'application SpringBoot.
 - Pour cela il suffit d'affecter le résultat de l'appel à la méthode run dans une variable de type ApplicationContext.
- La classe ApplicationContext propose une méthode getBean qui permet de récupérer une instance d'un composant Spring.

Attention, dans ce cours nous utiliserons rarement cette méthode. Le conteneur sera utilisée de manière transparente par injection de dépendance (notion que nous allons voir un peu plus tard)

```

package fr.eni.demo;

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

@SpringBootApplication
public class DemoSpringBeanApplication {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(DemoSpringBeanApplication.class, args);
    }
}

```

Définir un composant Spring avec @Component

- Créer une classe WelcomeController dans le package fr.eni.demo.controller, positionner l'annotation @Component

```

package fr.eni.demo.controller;

import org.springframework.stereotype.Component;

@Component
public class WelcomeController {

    public void welcome() {
        System.out.println("Bienvenue dans la démonstration de Container");
    }
}

```

- Par défaut le nom du bean est le nom de la classe dont la première lettre est mise en minuscule.
- Sur l'exemple, l'ajout de @Component permet de définir le bean nommé welcomeController

Injection d'un bean Spring par nom

- Compléter la classe d'exécution :

```

package fr.eni.demo;

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

import fr.eni.demo.controller.WelcomeController;

@SpringBootApplication
public class DemoSpringBeanApplication {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(DemoSpringBeanApplication.class, args);

        //Injection par nom
        WelcomeController welcomeCtrler = (WelcomeController) ctx.getBean("welcomeController");
        welcomeCtrler.welcome();
    }
}

```

L'utilisation de frameworks pour le développement avec Java EE

- La méthode `getBean()` permet l'instanciation du bean nommé « `welcomeController` » et d'affecter la référence de l'instance créée à la variable `welcomeCtrlr`. On dit que la référence est injectée par le conteneur dans la variable.
- La méthode `getBean` utilisée ici crée l'instance en cherchant un bean par son nom.

Il est possible de redéfinir le nom d'un bean.

- Une des possibilités, est de définir le nom en le passant en paramètre de l'annotation qui définit la classe comme étant un bean spring :

```
@Component("welcomeControllerBean")
public class WelcomeController {
```

- Dans ce cas on utilisera :

```
//Injection par nom
WelcomeController welcomeCtrlr = (WelcomeController) ctx.getBean("welcomeControllerBean");
welcomeCtrlr.welcome();
```

Injection d'un bean Spring par type

```
//Injection par type
WelcomeController welcomeCtrlr = ctx.getBean(WelcomeController.class);
welcomeCtrlr.welcome();
```

- Cette fois-ci, on recherche le bean en donnant le type : `WelcomeController.class`.
- Le bean `welcomeController` étant du type voulu va être instancié par Spring et injecté dans la variable `welcomeCtrlr`.

Bean de type singleton

- Par défaut, les beans sont de type singleton.
- Cela veut dire qu'à chaque demande d'injection, le conteneur fournira la même instance du bean :

```
// Singleton
WelcomeController welcomeCtrlr1 = ctx.getBean(WelcomeController.class);
System.out.println("welcomeCtrlr1=" + welcomeCtrlr1.toString());
WelcomeController welcomeCtrlr2 = ctx.getBean(WelcomeController.class);
System.out.println("welcomeCtrlr2=" + welcomeCtrlr2.toString());
```

- La console affiche :

```
welcomeCtrlr1=fr.eni.demo.controller.WelcomeController@6e0d4a8
welcomeCtrlr2=fr.eni.demo.controller.WelcomeController@6e0d4a8
```

- Les 2 appels à `getBean` ont bien retournés la même instance.

Bean de type prototype

- Il est possible d'affecter le type prototype à un bean.
- Dans ce cas, le conteneur crée une nouvelle instance à chaque injection.
 - Pour cela, nous ajoutons l'annotation @Scope sur le bean :

```
package fr.eni.demo.controller;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("prototype")
public class WelcomeController {

    public void welcome() {
        System.out.println("Bienvenue dans la démonstration de Container");
    }

}
```

- Avec cette nouvelle annotation, l'exécution de l'application affiche :

```
welcomeCtrler1=fr.eni.demo.controller.WelcomeController@1e8823d2
welcomeCtrler2=fr.eni.demo.controller.WelcomeController@c1a4620
```

- Les 2 appels à getBean ont bien retournés des instances différentes !