## Dependency injection in spring Boot

## @Component

@Component is an annotation that allows Spring to automatically detect our custom beans.

In other words, without having to write any explicit code, Spring will:

* Scan our application for classes annotated with @Component
* Instantiate them and inject any specified dependencies into them
* Inject them wherever needed

Exemple :

@Component  
public class Hello {  
 String name;  
 public Hello(String name){  
 this.name=name;  
 };  
 public Hello(){}  
 public void helloWorld()  
 {  
 System.*out*.println("hello world");  
 }  
 public void HelloHuman()  
 {  
 System.*out*.println("hello "+ name);  
 }  
}

@SpringBootApplication  
public class TestSpringApplication implements CommandLineRunner {  
 @Autowired  
 Hello h;   
 public static void main(String[] args) {  
 SpringApplication.*run*(TestSpringApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 h.helloWorld(); }  
}

## @Cofuguration and @Bean

La classe annotée avec l’annotation configuration permet de gérer l’injection de dépendance via des méthodes annotées @Bean.

@Configuration  
public class HelloConfig {  
 @Bean  
 Hello Hello(){  
 return new Hello();  
 }  
}

C’est le type de retour de la méthode qui décide l’objet à injecter. Ceci permet d’avoir un plus grand control sur la manière dont un objet de type Hello sera injecté, par exemple nous pouvons choisir le type de constructeur à utiliser dans l’injection :

@Configuration  
public class HelloConfig {  
 @Bean  
 Hello HelloHuman(){  
 return new Hello("Chifae");  
 }  
}

Changer le nom de la méthode ne fait aucune différence.

* ! à noter que l’on peut créer plusieurs méthodes d’injection du même type à condition de définir la méthode d’injection lors de la déclaration à l’aide d’un qualifier :

@Configuration  
public class HelloConfig {  
 @Bean  
 Hello Hello(){  
 return new Hello();  
 }  
 @Bean  
 Hello HelloHuman(){  
 return new Hello("Chifae");  
 }  
}

@SpringBootApplication  
public class TestSpringApplication implements CommandLineRunner {  
 @Qualifier("HelloHuman")  
@Autowired  
 Hello h;   
 public static void main(String[] args) {  
 SpringApplication.*run*(TestSpringApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 h.HelloHuman();  
 }  
}

* Affiche : hello Chifae
* Lorsque l’on declare un Bean à l’aide des deux méthode (methode 1 @configuration/@Bean et methode : @compnent, ceci sera considéré comme si l’on avait créé deux méthode avec l’annotaion @Bean dans une classe de configuration, un conflit sera présent et nous auront besoin d’utiliser un @Qualifier
* !! l’utilisation de @Autowired est déconseillé, il est conseiller d’initialiser les composant dans un constructeur ou dans un setter :

public class HelloService {  
 Hello hello;  
 public HelloService(Hello hello){  
 this.hello=hello;  
 }  
  
 public Hello getHello() {  
 return hello;  
 }  
}

@Configuration  
public class HelloConfig {  
 @Bean  
 Hello Hello(){  
 return new Hello();  
 }  
 @Bean  
 Hello HelloHuman(){  
 return new Hello("Chifae");  
 }  
 @Bean  
 HelloService helloService()  
 {  
 return new HelloService(Hello());  
 }  
}

@SpringBootApplication  
public class TestSpringApplication implements CommandLineRunner {  
 @Qualifier("HelloHuman")  
@Autowired  
 Hello h;  
 @Autowired  
 HelloService service;  
 public static void main(String[] args) {  
 SpringApplication.*run*(TestSpringApplication.class, args);  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 service.getHello().helloWorld();  
 }  
}