REST / Spring Boot

-L’inversion de contrôle avec Spring Boot-

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Objectifs du chapitre

- Les objectifs de ce chapitre sont:
- Prise de connaissance du framework Spring.
- Créer un projet Spring rapidement avec Spring Boot.
- Manipuler l'IoC avec Spring.

Le framework Spring

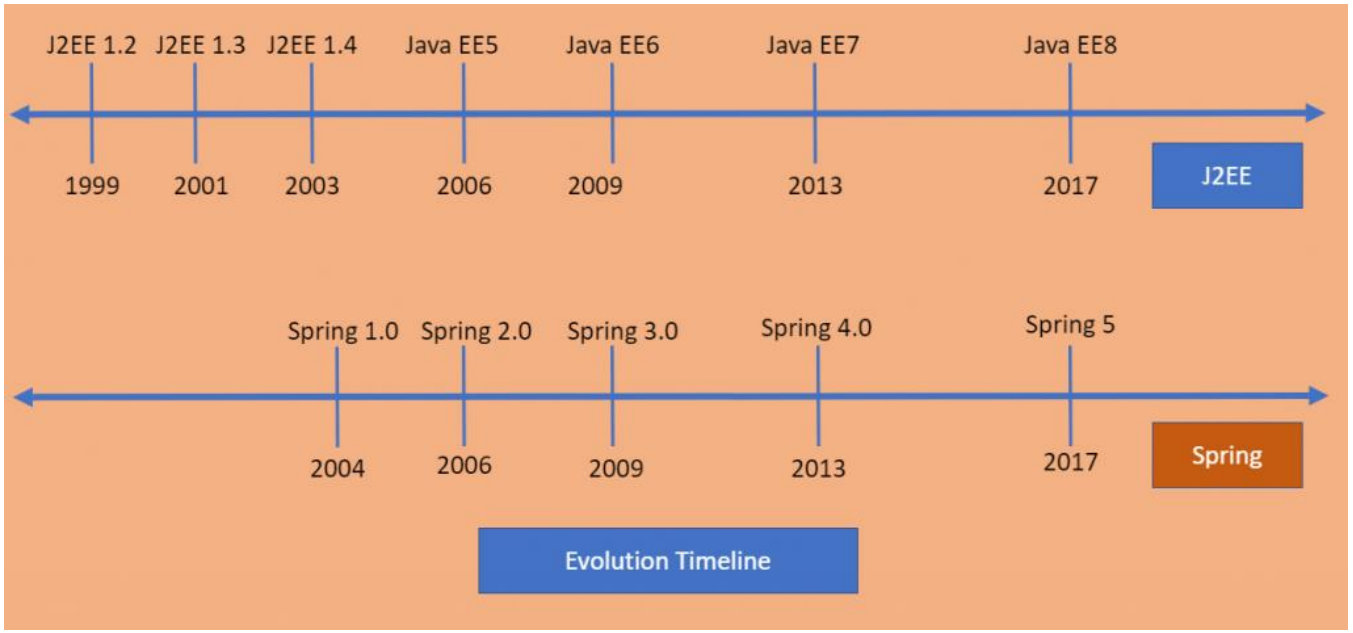
- **Framework open-source** créé en 2002 par Rod Johnson,
  avec pour objectif de faciliter le développement d'applications Java et Java EE.
- Le projet est supporté par la société SpringSource, rachetée en 2009 par VMware.

Le framework Spring

- Spring est un **conteneur léger** : il offre une infrastructure similaire à un conteneur « lourd » tel qu'un serveur d'application Java EE (gestion du cycle de vie des objets, gestion des transactions, …)
- Avantages d'utiliser Spring:
  - Il évite le *plumbing code* qui a fait la mauvaise réputation de Java EE, non intrusif, rapide à démarrer, utilisable dans plusieurs contextes (application web, application standalone, …).
  - Encourage et facilite les « meilleurs pratiques » de programmation : architecture modulaire, tests unitaires, …
  - S'intègre aux différents frameworks existants : Struts, Hibernate, …
  - Suffisamment modulaire que pour être introduit progressivement dans un projet existant.

Le framework Spring

- Concordance des dates de sorties entre les versions Java EE et Spring, chacun enrichit mutuellement l'autre.

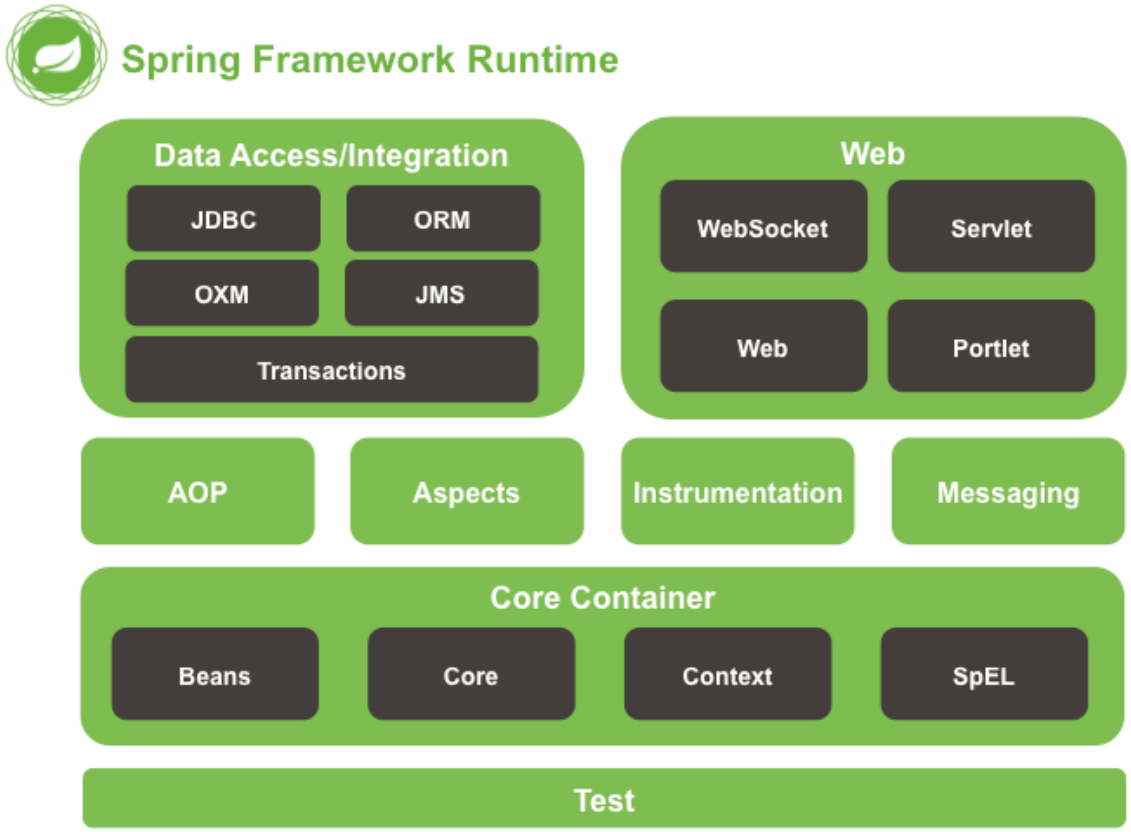


technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Le framework Spring

- Spring est constitué de 20 modules regroupés comme ci-dessous. Le module Core est celui qui nous concerne pour l'IoC



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Le framework Spring

- Liste non exhaustive des modules Spring:
  - **Spring Core** : contient le conteneur léger ; on y retrouve également les mécanismes d'injection de dépendances et d'inversion de contrôle ;
  - **Spring AOP** : module de programmation orientée aspect ;
  - **Spring DAO** : fournit un mécanisme d'abstraction de l'accès aux données (JDBC, LDAP, …) ainsi que la gestion des transactions ;
  - **Spring ORM** : intègre les solutions de mapping objet-relationnel, (Hibernate, JPA, iBatis, …) au framework ;
  - **Spring Web** : contient Spring MVC ainsi que des modules d'intégration des frameworks Web existants (Struts, …) ;
  - **Spring Security** : regroupe des composants permettant de sécuriser une application Java EE ;

Le framework Spring

- La configuration de l'IoC se fait soit par XML, annotations ou code classique. L'injection par annotation est effectuée avant l'injection XML , donc cette dernière redéfinit les annotations. A l'inverse des EJB, les beans Spring sont utilisable non seulement dans java EE mais aussi dans une simple application console.
- Sans le IoC container les beans sont juste des classes et les fichiers XML sont inutiles.
- Il existe plusieurs IoC container, entre autres le *servlet container,* pour les beans de web services et le *jdbc container* en lien avec les bases de données.

Spring Boot

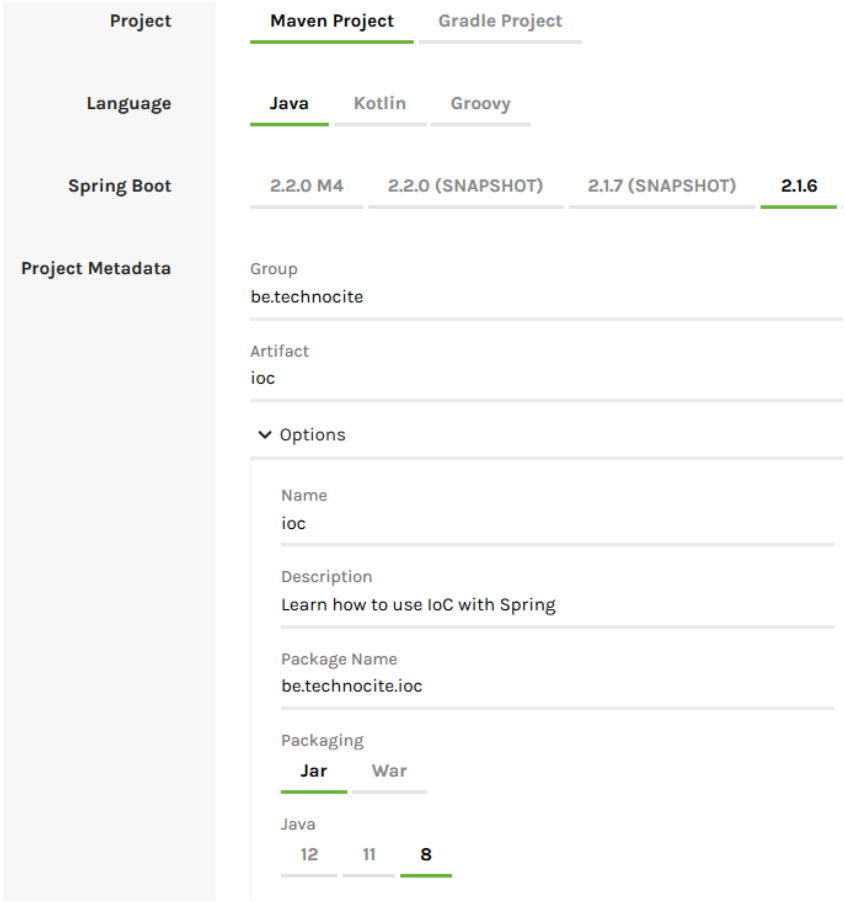
- Généralement plusieurs modules Spring coexistent dans un projet. Mais mettre sur pied l'armature d'un microservice qui inclus plusieurs modules peut s'avérer chronophage.
- En 2014 le framework Spring Boot est né. Il facilite la mise en place de l'environnement d'une application Spring stand alone, afin de pouvoir immédiatement se concentrer sur le *buisness code*.

Spring Boot

- Les avantages de Spring Boot sont :
  - Server Tomcat ou Jetty embarqué directement (pas besoin de déployer des fichiers WAR)
  - Fournit des *starters* qui sont dépendances optionnelles pour simplifier la configuration du *build*
  - Configuration automatique de Spring et des librairies tiers dès que possible en fonction des dépendances
  - Des fonctionnalités de production déjà prêtes comme le monitoring et l'externalisation de l'application
  - Aucune configuration XML requise

Création d'un projet Spring Boot

- Aller sur le site de Spring Initializr https://start.spring.io/
- Paramétrages:



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Création d'un projet Spring Boot

- Après avoir importé le projet Maven généré, on peut observer:
  - pom.xml : la configuration du projet Maven
  - mvnw.cmd et mvmw: des scripts pour initialiser automatiquement l'environnement pour Maven
  - Le dossier .mwn/wrapper: contient le nécessaire en relation avec les scripts précédents
  - Le dossier resources: emplacement des fichiers utilisés liés à l'exécution d'une application Java, ils seront contenu dans le .jar
- Certains éléments peuvent être supprimés car ils sont en dehors du scope de ce chapitre:
  - .gitignore: fichiers qui ne doivent pas êtres commités par le VCS Git
  - Le dossier test pour tester notre application
  - HELP.md, un fichier texte pour nous guider avec Maven

Création d'un projet Spring Boot

- Décryptons la classe de notre application Spring générée:

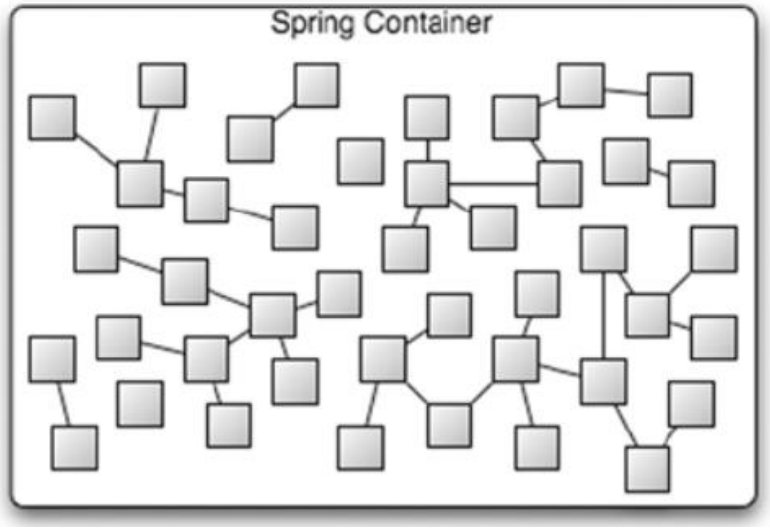
```
@SpringBootApplication
public class IocApplication {

  public static void main(String[] args) {
    SpringApplication.run(IocApplication.class, args);
  }

}
```

- Elle lance la classe *SpringApplication*, responsable du démarrage de l'application Spring. Cette classe va créer un *ApplicationContext* dans lequel iront toutes les configurations générées automatiquement ou ajoutées par vos soins.

Création d'un projet Spring Boot

Context

- Qu'est ce que l' ***ApplicationContext*** ?
- Comme dit précédemment il existe plusieurs containers de beans. Ils sont caractérisés en 2 types distincts, les Bean Factories et les Applications Contexts.



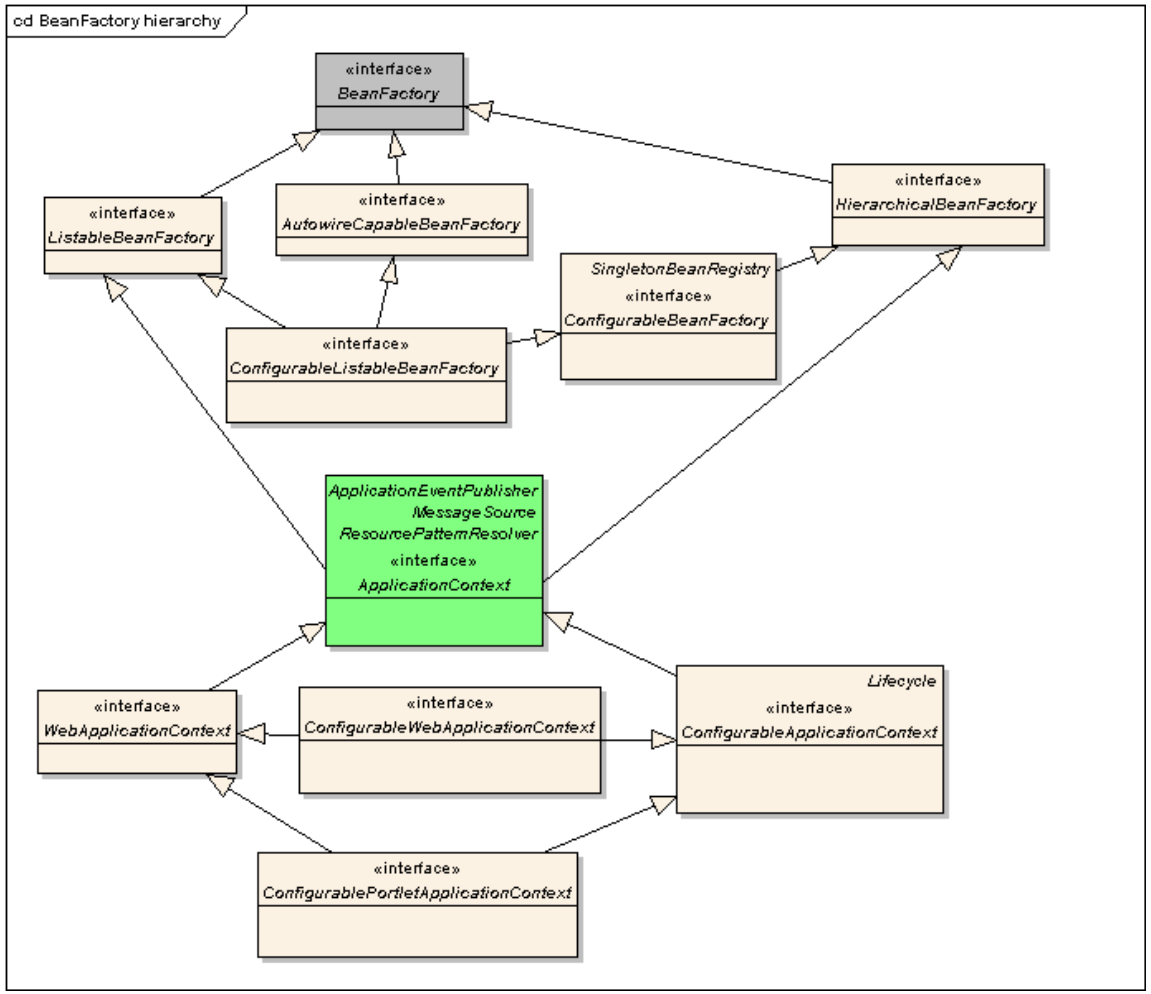
Création d'un projet Spring Boot

Context

- Bean Factories: simples container qui créent les beans et les lie entre eux.
- Application contexts: supportent quelques fonctionnalités supplémentaires
  - Résolution des messages pour l'internationalisation
  - Charger génériquement les ressources comme les images
  - Publication d'évènement vers les beans qui ont été renseignés comme *listeners*
- Le BeanFactory est préféré quand les ressources sont limités comme dans un smartphone.

Création d'un projet Spring Boot

Context



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Création d'un projet Spring Boot

- L'annotation***@SpringBootApplication***, qui est une simple encapsulation de trois annotations :
- **@Configuration** : donne à la classe actuelle la possibilité de définir un groupe de beans configurés pour un type de comportement de notre application. Il peut donc y en avoir plusieurs par projet.
- **@EnableAutoConfiguration** : l'annotation vue précédemment qui permet, au démarrage de Spring, de générer automatiquement les configurations nécessaires en fonction des dépendances situées dans notre *classpath*.
- **@ComponentScan** : Indique qu'il faut scanner les classes de ce package afin de trouver des Beans de configuration.

Exemple de DI

- Exemple d'injection avec Spring dans le cadre d'une application console.
- Il faut au préalable ajoute la ligne suivante dans le fichier application.properties: spring.main.web-environment=false
- Nous ne pouvons accéder à notre propriété *helloWorldService* depuis le main static. La solution est de passer par l'interface *CommandLineRunner* de SpringBoot.
- ***@Component*** déclare la classe comme étant un bean, il sera ajouté dans le container de l'application context.
- Alternativement si on veut le configurer pour divers scénario il aurait fallu le déclarer dans ***@Configuration.***

Exemple de DI

```java
@SpringBootApplication
public class IocApplication implements CommandLineRunner {

  @Autowired
  private HelloWorldService helloWorldService;

  public static void main(String[] args) {
    SpringApplication app = new SpringApplication(IocApplication.class);
    app.setBannerMode(Banner.Mode.OFF);
    app.run(args);
  }

  @Override
  public void run(String... args) throws Exception {
    helloWorldService.showMessage();
  }
}
```

```java
@Component
public class HelloWorldService {

    public void showMessage(){
        System.out.println("Hello");
    }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Exercice

- Nous avons une classe *BookShelf* qui contient des livres et fait appel à une classe *XMLBookImporter* pour importer les livres et se les assigner. Réécrivez la méthode *toString* pour afficher le contenu de l'étagère à livres.
- Ordre des dépendances:
  - *IoCApplication* se fait injecter *BookShelf*
  - *BookShelf* se fait injecter un *IBookImporter*

@Autowire

- L’injection peut se faire par propriété, constructeur ou setter.

Pa constructeur	Par setter	Par propriété
quand des bean sont essentiels pour le fonctionnement de notre classe.	pour les bean qui sont optionnels.	plus simple en alternative

@Configuration

- Il existe deux types de beans, les ordinaires et les factory beans. Ces derniers sont utilisés pour instancier et configurer les premiers.
- Les classes de configuration vont donc elles déclarer les bean disponibles. Plus besoin de *@Component*.
- On passe par un bean factory quand on doit paramétrer des bean complexes, en fonction de l'environnement par exemple.
- On garde toujours le *@Autowired* sur la classe qui doit recevoir l'injection. Nous gardons aussi *@ComponentScan* car cette annotation recherche aussi les classes de configuration.

Exemple de DI

Par ApplicationContext	Par bean factory
@Component sur la classe du bean qui s'injecte	@Bean sur la méthode qui renvoit l'injection
@Autowired sur la propriété du bean qui reçoit l'injection	@Autowired sur la propriété du bean qui reçoit l'injection
@ComponentScan necessaire sur une classe du package (généralement le main)	@Configuration sur la classe qui déclare des bean
@Component sur le classe du bean qui reçoit une injection dans son corps	Aucune annotation sur la classe qui est un bean

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

@Configuration

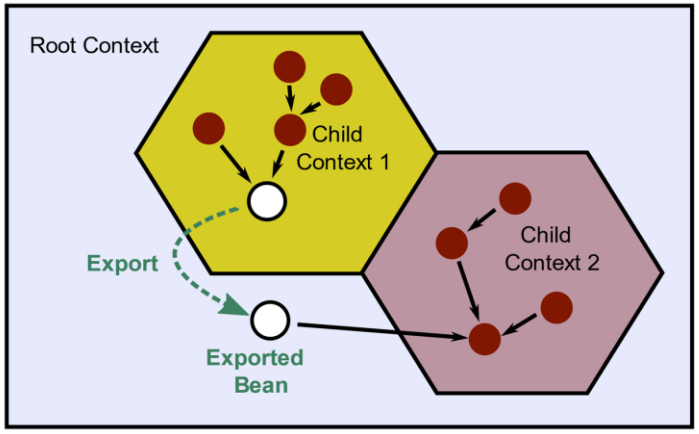
```java
@Configuration
public class CarConfiguration {

  @Bean
  public MyCarFactoryBean carFactoryBean(){
  MyCarFactoryBean cfb = new MyCarFactoryBean();
  cfb.setMake("Honda");
  cfb.setYear(1984);
  return cfb;
  }

  @Bean
  public Person aPerson(){
  Person person = new Person();
  person.setCar( carFactoryBean().getObject());
  return person;
  }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

@Configuration

- Chaque classe avec l'annotation @Configuration créera un context enfant dans l'*Application Context.*
- Les contexts enfants peuvent accéder aux bean de leur contexte parent et non l'inverse.
- De part l'annotation *@SpringBootApplication* tous les bean définit dans les context seront enregistré dans l' *Application Context* ne rendant ainsi pas notre application modulaire.



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

@Configuration

- Pour modulariser l'application il faut soit s'amuser avec les classes avancées de Spring (tutoriel).
- Soit modulariser notre application via Maven.
- **Exercice**: déclarer tous les bean du projet *BookShelf* via une classe de configuration. Créez les beans individuellement.

@Qualifier

- Par défaut Spring résout les types à injecter par le type de classe. Si plus d'un type sont disponible l'exception *NoUniqueBeanDefinitionException* est lancée.
- Pour le résoudre on ajoute à *@Autowired* l'annotation *@Qualifier* en précisant le nom du bean écrit dans *@Component*
- Autre solution, utiliser *@Primary* sur la classe du bean à favoriser
- **Exercice**: Reprendre l'exercice d'injection par ApplicationContext et ajouter un *PDFBookImporter* et un *CSVBookImporter.* Résoudre les conflits d'injection.

@Order

- Quand une classe souhaite se voir injecter toutes les implémentations d'une interface, on injecte une collection de type de l'interface. Spring s'occupera de créer et injecter la liste de toutes les implémentations.
- Il est possible d'ordonner les implémentations dans la liste avec *@Order* au dessus de la déclaration de classe des bean qui implémentent l'interface.
- Cela fonctionne aussi avec l'héritage des classes concrètes / abstraites.
- L'index N°0 peut être forcé via le paramètre d'annotation *Ordered. HIGHEST_PRECEDENCE.*
- **Exercice**: Injecter une liste de *IBookImporter* dans le *BookShelf*. Ordonnez les selon votre ordre de préférence.

@Scope

- Un bean peut avoir une scope (étendue) de type singleton ou de type prototype, elle est précisée avec ***@Scope("prototype ").***
- **Singleton** (par défaut): A la première injection le bean est instancié, toutes les demandes suivantes d'injections retournent la même instanciation du bean. En somme il n'y a qu'une instanciation de se bean pour toute la vie de l'application. Exemple: Une connexion à une DB.
- **Prototype**: Une instanciation différente est faite à chaque demande d'injection du bean. Exemple: Un validateur de formulaire.
- **Exercice**: Créer une classe A et B qui reçoivent un *BookShelf* en singleton, afficher les adresses mémoires afin de vérifier si c'est bien la même instanciation. Faites de même avec le scope prototype.

@Scope

- Il peut arriver qu'un bean singleton se voit injecter un bean d'un scope plus étroit, comme un prototype.
- Dès lors le prototype ne sera instancié qu'une fois en même temps que le singleton. C'est ce que l'on appelle **the scoped bean injection problem**.
- Pour le résoudre il faut passer des arguments à l'annotation :

```
@Scope(value=BeanDefinition.SCOPE_PROTOTYPE, proxyMode=ScopedProxyMode.TARGET_CLASS)
```

- Spring utilise le pattern proxy. C'est le proxy du prototype qui est injecté et à chaque fois qu'une méthode du proxy est appellée, elle redirigera l'appel vers la vraie classe prototype.

@PostConstruct

- Une méthode permet d'être notifé juste après que les propriétés de notre bean aient été initialisées.
- Il faut rajouter *@PostConstruct* au dessus de la méthode désirée de notre bean à surveiller.
- Utile quand on veut initialiser les variables d'une classe qui n'a pas de constructeur, ou bien pour populer une base de donnée durant notre développement.

@PostConstruct

```java
@Component
public class DbInit {

    @Autowired
    private UserRepository userRepository;

    @PostConstruct
    private void postConstruct() {
        User admin = new User("admin", "admin password");
        User normalUser = new User("user", "user password");
        userRepository.save(admin, normalUser);
    }
}
```

- Dans l'exemple, Spring va d'abord initialiser *UserRepository* puis appeler la méthode annotée.

@PreDestroy

- La méthode annotée de *@PreDestroy* sera appelée juste avant que Spring ne supprime le bean de *l'application context.*
- Utile pour nettoyer une classe, fermer les connexions et streams.

```
@Component
public class UserRepository {

    private DbConnection dbConnection;
    @PreDestroy
    public void preDestroy() {
        dbConnection.close();
    }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Evènements

- Spring nous permet d'être abonné à des évènements de son fonctionnement, par exemple l' Application Context va lancer un évènement une fois qu'il a démarré, s'est arrêté, se rafraichit…
- Nous pouvons créer notre évènements personnalisés synchrones ou asynchrones. Cela trouve son intérêt quand on veut échanger des informations entre des composants pauvrement couplés ou éloignés.

@Value

- Selon l'environnement notre application se doit d'être paramétrée différemment. Faire une configuration via du code n'est pas un bon choix car il faudrait réexporter notre application à chaque changement d'environnement.
- Passer par un fichier texte facilement éditable est plus adapté, on utilise avec Spring le fichier application.properties, ses valeurs sont récupérées via l'annotation @Value. La chaine de caractères passée utilise les Expressions Language (SpEL).

```
@Value("${value.from.file}")
private String valueFromFile;
```

- Nous trouverons dans ce fichier les identifiants de connexion à une DB, les ports et url pour contacter d'autres web services…

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les exceptions courantes

- ***UnsatisfiedDependencyException*** **:** Quand Spring n'arrive pas à résoudre une dépendance à injecter (package non scanné, plusieurs candidats, bean non définit, …)

- ***BeanCurrentlyInCreationException*** : Généralement levée quand il y a une dépendance circulaire entre plusieurs beans.
  - Bean A → Bean B → Bean A
  - Bean A → Bean B → Bean C → Bean D → Bean E → Bean A

- ***NoSuchBeanDefinitionException*** : Quand un bean n'est simplement pas définit dans le context Spring.

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Exercice

- Exercice: Détecter naissance de notre application et afficher à ce moment la phrase « Asta la Vista » + prénom de l'utilisateur entré dans le fichier des propriétés.

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions