

Essentiel Spring Boot

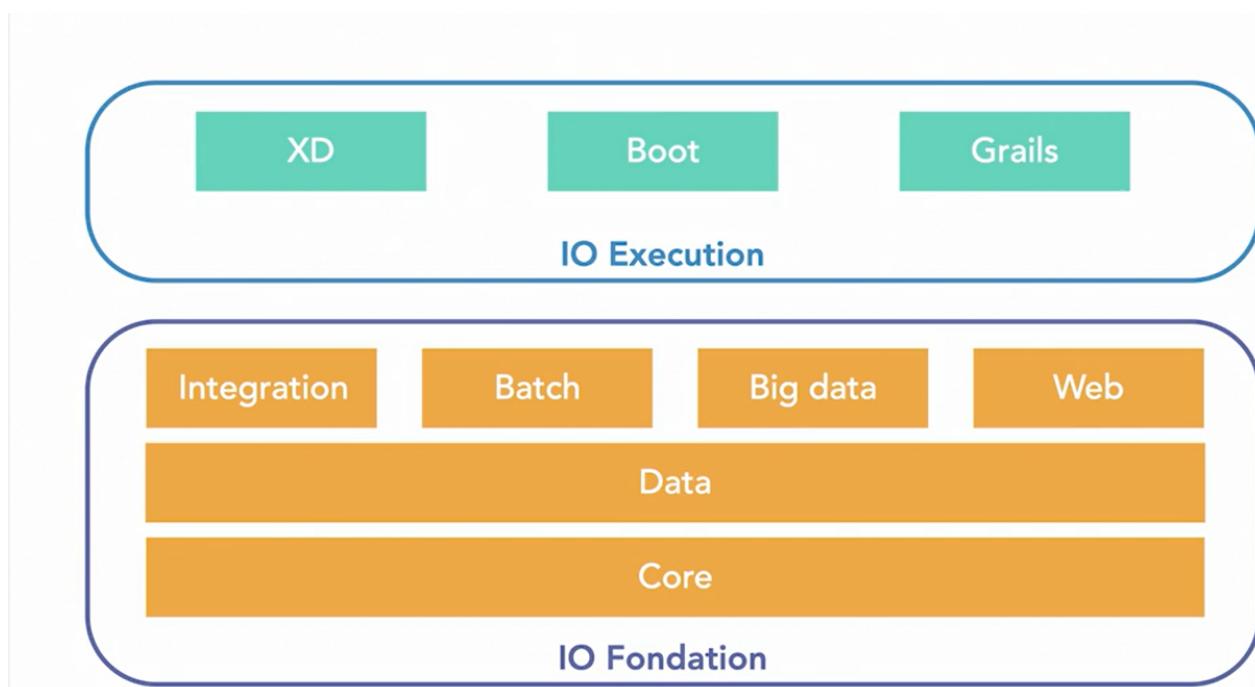
Auteur: CAMARA Laby Damaro

sources:

1. https://github.com/camara94/essentiel_spring_boot
2. [linkedin Learning](#)
3. [Openclassrooms](#)

Dans cours PDF,nous allons découvrir ensemble le framework Spring et de son module Spring Boot, pour le développement d'applications Java. À l'aide d'exemples pratiques, nous verrons comment injecter des dépendances puis nous verrons comment lancer une application Spring. nous aborderons le principe d'autoconfiguration et nous apprendrons l'utilisation utilisation des **starters** et **différents types de contrôleurs**. Nous testerons et nous superviserons nos applications avec **JUnit** et **Spring Boot Actuator**, avant d'exécuter des beans au démarrage. À la fin de cette formation, nous aurons acquis toutes les bases pour mettre en œuvre Spring Boot.

Le Noyau de Spring



Définition de Spring Boot

Spring Framework et Spring Boot ! Spring Framework et Spring Boot offrent un environnement de développement solide et efficace, qui va vous simplifier la vie. La gestion des dépendances, la configuration, la gestion des propriétés et le déploiement seront bientôt des jeux d'enfants !

Spring Boot

- Module de Spring
- Point d'entrée à la stack Spring
- Facilite le démarrage d'un projet Spring

Les Avantages d'utiliser Spring Boot

Avantages

- Générateur d'applications Spring
- Autoconfiguration
- Starters : gestion des dépendances
- Plus besoin de spécifier les versions des dépendances
- Monitoring : Actuator

Les Inversion de Contrôle en Spring Boot

Inversion de contrôle (IoC)

- Patron d'architecture
- Donne le contrôle à une entité externe
- Exemple : Tomcat (conteneur de servlets)
- Change la façon d'écrire du code

Les Avantages de l'Inversion de Contrôle

Avantages

- Code plus clair
- Réduit la complexité et les sources de bug
- Couplage faible
- Améliore la maintenabilité
- Testabilité

L'Inversion de Contrôle en Spring Boot avec les Beans

Conteneur IoC de Spring

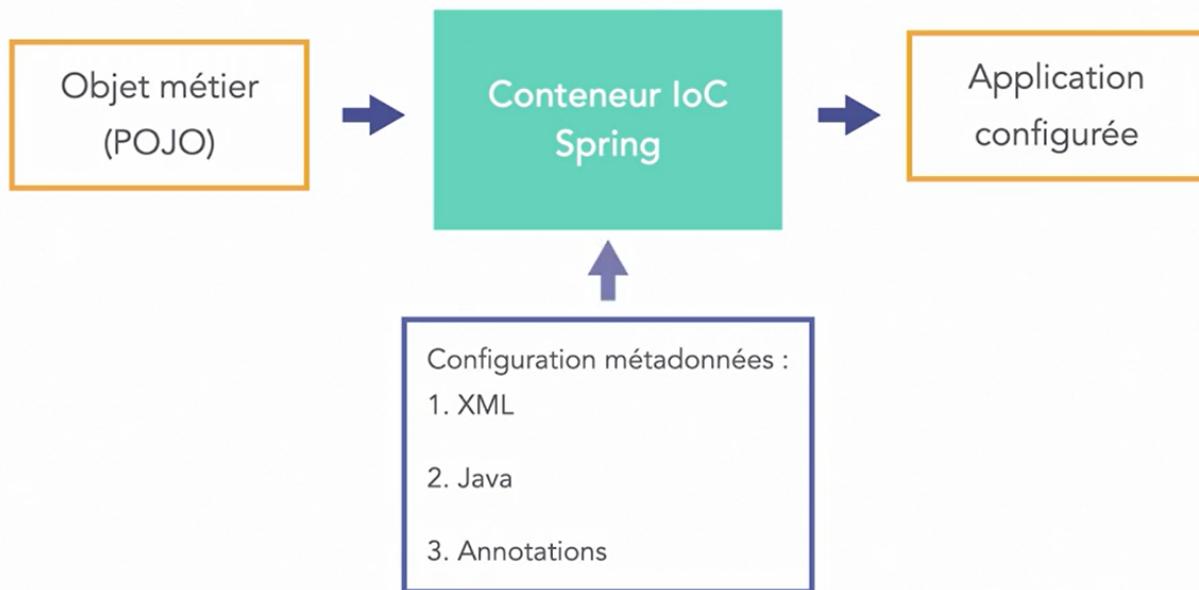
- Gère l'instanciation des objets (beans)
- Injection des dépendances entre les beans
- Gestion du cycle de vie des beans
- Configuration établie une seule fois

Le conteneur IOC de Spring Boot (ApplicationContext)

ApplicationContext

- Élément central de Spring framework
- Conteneur IoC de Spring
- Encapsule BeanFactory
- Crée les beans en se basant sur la configuration des métadonnées

Le Fonctionnement du Conteneur IOC de Spring Boot



LinkedIn Learning

Illustration De L'IOC en Spring

Ici nous allons créer quatre classe(A, B, C, Main) pour illustrer l'inversion de contrôl avec spring boot.

Avant de créer les classes, il faut ajouter cette dépendance la section **dependances** dans le fichier **pom.xml** dans un projet **maven** si ce n'est pas directement un projet spring par default.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.8</version>
</dependency>
```

Classe A

```
package com.camaratek;
import org.springframework.stereotype.Component;
@Component
public class A {
    public A() {
        System.out.println( "Création d'Objet avec spring IOC Annotation A" );
    }
}
```

Classe B

```
package com.camaratek;
import org.springframework.stereotype.Service;
@Service
public class B {
    public B() {
        System.out.println( "Création d'Objet avec spring IOC Annotation B" );
    }
}
```

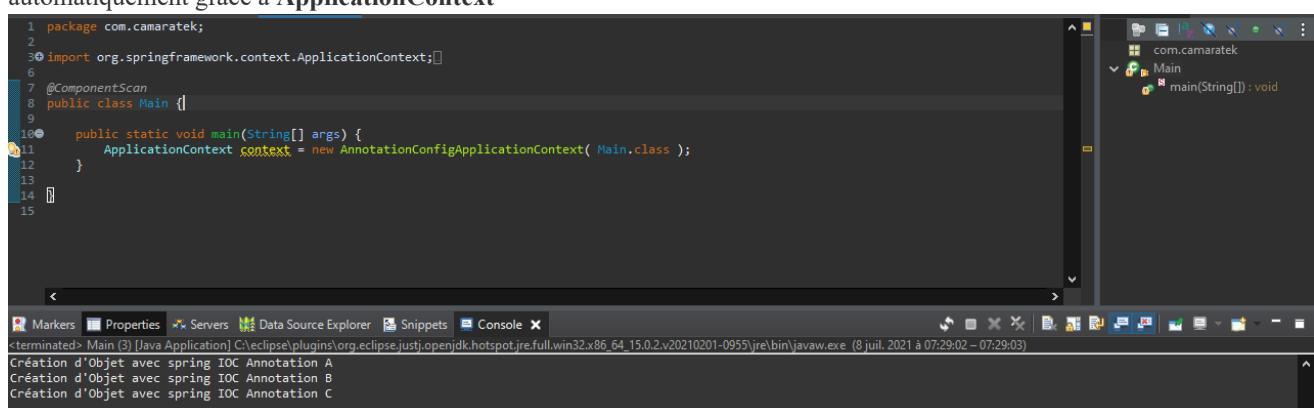
Classe C

```
package com.camaratek;
import org.springframework.stereotype.Repository;
@Repository
public class C {
    public C(A a, B b) {
        System.out.println( "Création d'Objet avec spring IOC Annotation C" );
    }
}
```

Classe Main

L' **ApplicationContext** est l'interface centrale au sein d'une application Spring qui est utilisée pour fournir des informations de configuration à l'application. Il implémente l'interface BeanFactory. Par conséquent, ApplicationContext inclut toutes les fonctionnalités de BeanFactory.

C'est pourquoi ici, ici nous n'avons pas besoin faire des instanciation avec le mot clés **new**, et **spring boot** s'en charge automatiquement grâce à **ApplicationContext**



```
1 package com.camaratek;
2
3 import org.springframework.context.ApplicationContext;
4
5 @ComponentScan
6 public class Main {
7
8     public static void main(String[] args) {
9         ApplicationContext context = new AnnotationConfigApplicationContext( Main.class );
10    }
11
12 }
13
14
15
```

The screenshot shows the Eclipse IDE interface. The code editor displays the Main.java file with the provided code. To the right, the Java Explorer view shows the project structure with a Main class under the com.camaratek package. The status bar at the bottom indicates the current date and time: "Création d'Objet avec spring IOC Annotation A" and "Création d'Objet avec spring IOC Annotation B".

La Configuration des Métadonnées en Spring Boot

En Spring Boot, on peut procéder la configuration de trois manières à savoir:

Configuration des métadonnées

- XML
- Configuration Java
- Annotations Spring

La Configuration avec les Classe Java

Pour cela, nous utilisons le plus souvent les annotations suivantes:

1. @Configuration: cette annotation indique à **Spring boot** qu'il s'agit d'une classe de configuration.
2. @Bean: celle ci indique précise une méthode qui crée des objets Java
3. @Autowired elle permet de faire les **Injection de dependance** en

Configuration Java

- @Configuration
- @Bean
- @Autowired

```
1 package com.example.demo.autoconfigure;
2
3 import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
4
5 @Configuration
6 @ConditionalOnClass(LogService.class)
7 @EnableConfigurationProperties( LogProperties.class )
8 public class LoggerAutoConfiguration {
9
10     @Bean
11     @ConditionalOnMissingBean
12     public LogService logService() {
13         return new LogServiceImp();
14     }
15 }
```

La Configuration avec les Annotation Spring Boot

1. @Component: elle marque une classe java en tant que Bean pour les mécanismes d'analyse spring puis l'ajouter au contexte de l'application. elle a plusieurs dérivées: **@Repository, @Service, @Controller**

2. **@Repository**: qui est utilisée sur les classes(Interfaces) java qui manipulent ou accèdent directement aux bases de données.
3. **@Service**: elle marque une classe java qui effectue des traitements métiers.
4. **@Controller**: est utilisée pour indiquer que la classe est un controller **Spring Boot**
5. **@ComponentScan**: qui est utilisée pour indiquer à **Spring Boot** les **packages java** qu'il faut utilisés pour trouver les composants **Spring**

Annotations Spring

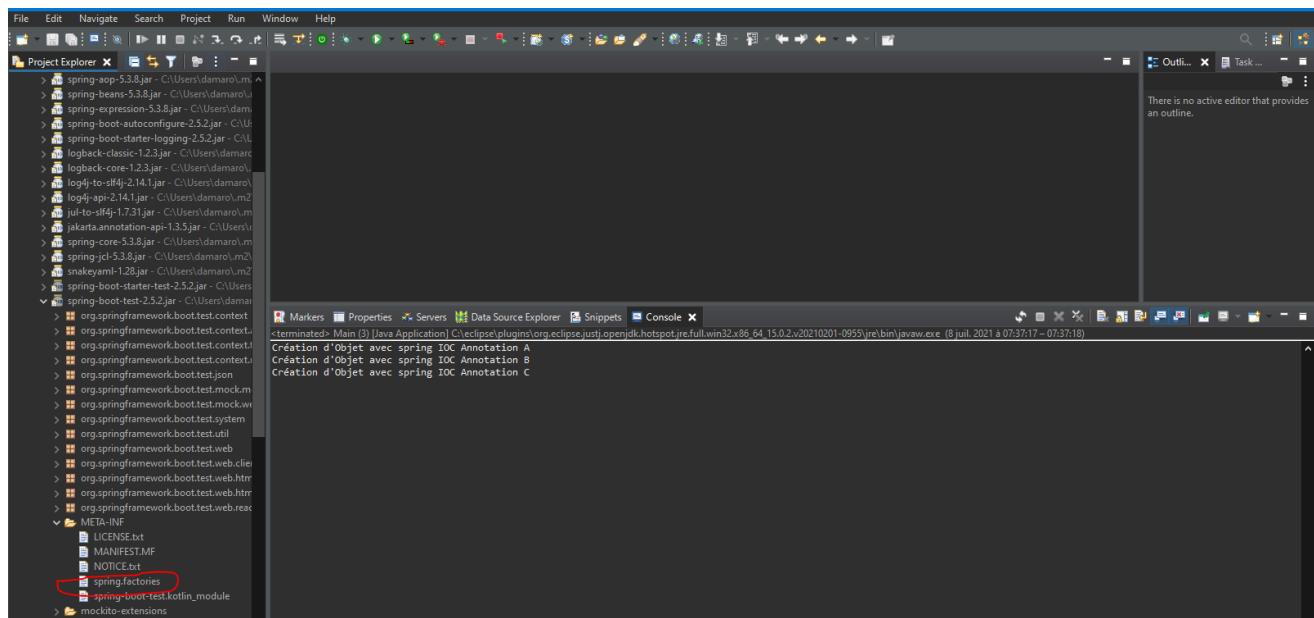
- Désigner un bean : annotation **@Component**
- Stéréotypes **@Repository**, **@Service**, **@Controller**
- **@ComponentScan**

Comment créer une Application Spring Boot

La façon la plus simple est d'utiliser le site [Spring Initializr](#)

Spring Initializr

- start.spring.io
- Générer un projet Spring Boot
- Permet de démarrer rapidement un projet Spring Boot



Comment Proceder avec Spring Initializr

Le formulaire de spring initializr nous permet choisir:

- le type de projet: **Maven, Gradle**
- le language à utiliser: **Java, Kotlin, Groovy**
- la version de spring boot à utiliser
- le nom du package qu'on veut utiliser: (group)
- le nom du projet: (artifact)
- le packaging: **jar ou war**
- les dependances qu'on souhaite ajouter à notre

Lorqu'on clique sur le bouton **générate** il va enrégistrer le projet dans notre PC sous format zip

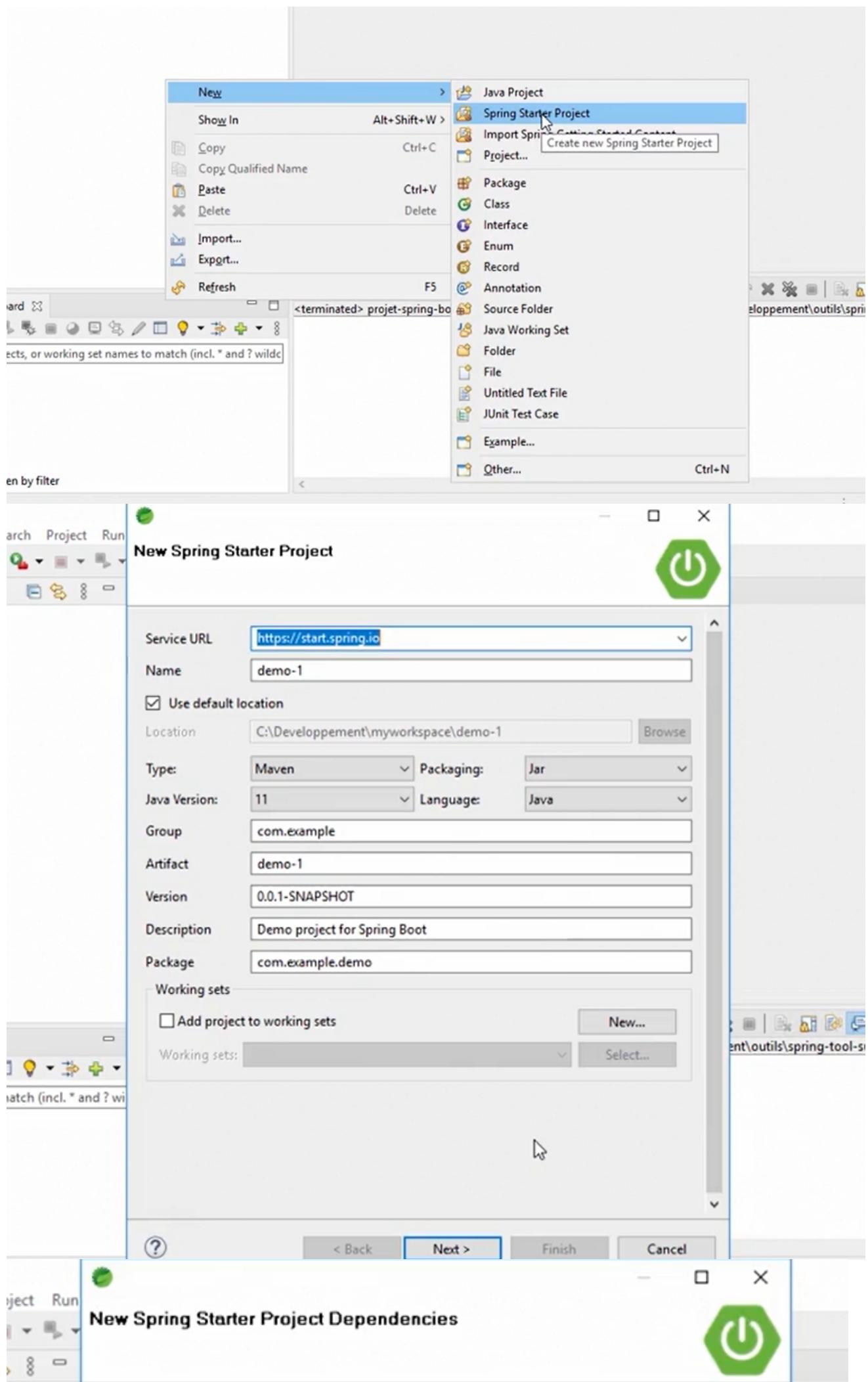
The screenshots show the Spring Initializr configuration interface. Both instances have the following settings:

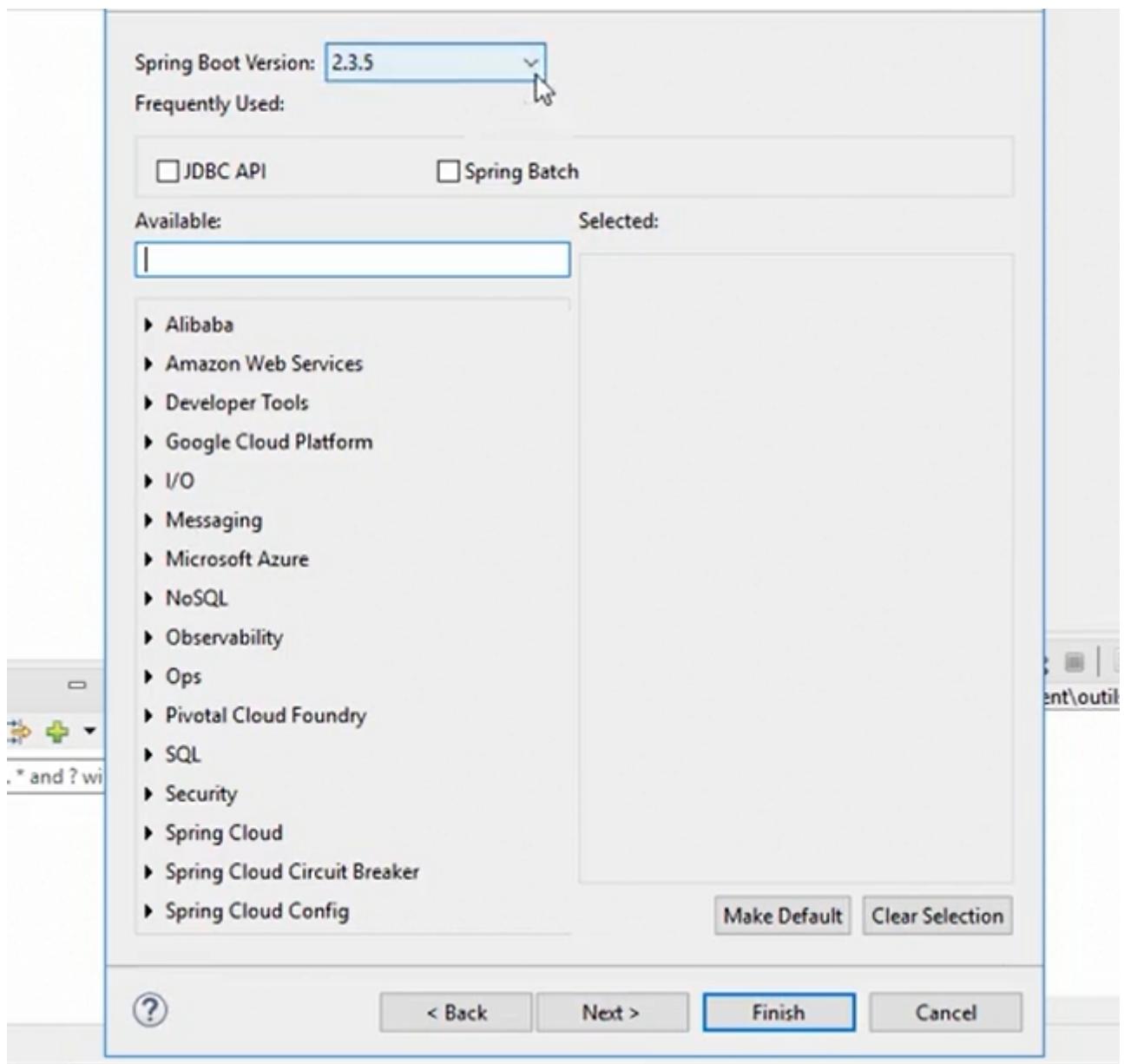
- Project:** Maven Project (selected)
- Language:** Java (selected)
- Spring Boot:** 2.3.5 (selected)
- Project Metadata:**
 - Group: com.example
 - Artifact: demo
- Dependencies:** No dependency selected (in the first screenshot), or one dependency listed (in the second screenshot).

Comment Proceder Avec Les IDE

1. on peut utiliser IDE STS
2. on peut aussi utiliser IDE Eclise avec le plugin STS
3. on peut utiliser IDE Intelij

et la procedure est la même que celle qu'on a utilise precedemment avec **Spring Initializr**, quelques captures d'écran pourraient illustrer mon propos





Une Dépendance Starter Spring Boot

Une **Dependance Starter Spring** est une qui est une dépendance qui lui même contenant plusieurs dépendances **retrocompatibles**.

L'un des points forts de Spring Boot est qu'il trouve lui même les dépendances compatibles entre elles.

Quelques Fichiers de Configurations

1. l'un des fichiers imports est **pom.xml** qui permet de:

- lister toutes les dépendances d'une application **Spring Boot**
- ajouter d'autres dépendances à notre application
- modifier le nom, la description
- configurer le build, les propriétés

```

...
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/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>2.5.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>SpringWeb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>SpringWeb</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>11</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.thymeleaf</groupId>
      <artifactId>thymeleaf</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        ...

```

2. nous avons aussi le fichier **application.properties** qu'on peut modifier son extension en **yaml** si nous le souhaitons bien, ce fichier permet également de faire des configuration aussi telles que:

- changement de port
- configuration de base de données
- ...

```

application.properties
1 debug=true
2 log.nom=CAMARA
3 log.prenom=Laby Damaro
4 log.age=28
5 log.status=Etudiant
6 log.niveau=en mastère pro business intelligence
7 log.universite=ISET Radès
8 log.ville=Tunis
9 log.pays=Tunisie
10 server.port=8083

```

AutoConfiguration En Spring Boot

Autoconfiguration

- Configuration automatique en fonction des jars dans le classpath
- `@EnableAutoConfiguration` : active l'autoconfiguration
- Si l'utilisateur n'a pas défini de configuration

Classe d'autoconfiguration

- Se base sur les deux annotations Spring
- `@Configuration` : classe de configuration
- `@Conditional` : depuis Spring 4

Fonctionnement

- Autoconfiguration Spring Boot : META-INF/spring.factories du jar `spring-boot-autoconfigure`
- Définition de la liste des classes d'autoconfiguration
- `@AutoConfigureBefore` et `@AutoconfigureAfter`

Connaître Les AutoConfigurations en Spring Boot

Conditions d'autoconfiguration

- Classes
- Beans
- Propriétés
- Application Spring
- Ressources

Exemples

- `@ConditionalOnMissingClass`
- `@ConditionalOnClass`
- `@ConditionalOnMissingBean`
- `@ConditionalOnBean`
- `@ConditionalOnWebApplication`
- `@ConditionalOnNotWebApplication`

© OpenClassroom - 2018

Starter Spring Boot

Starters Spring Boot

- POM standard : définit des dépendances pratiques vers des librairies
- Beaucoup de possibilités : batch, intégration, web, test
- `spring-boot-starter-*`

Quelques Starter Spring Boot

Quelques starters

- Spring-boot-starter
- Spring-boot-starter-web
- Spring-boot-starter-data-jpa
- Spring-boot-starter-test
- Spring-boot-starter-tomcat

Le Modele MVC En Spring Boot

Modèle-Vue-Contrôleur (MVC)

- Design pattern du développement web
- Séparation du code en trois parties

Modèle : données, dynamique

Vue : statique, affichage

Contrôleur : fournit les données à la vue, business

Spring MVC

MVC Spring

- Spring-boot-starter-web
- Web MVC
- Services RESTful

Contôller en Spring Boot

Contrôleur

- Web MVC : @Controller
- Services RESTful : @RestController

CommandLineRunner

CommandLineRunner

- Interface Spring Boot
- Implémenter une méthode run()
- Run() sera exécutée après le chargement du contexte Spring et avant la fin de l'exécution de la méthode main()
- Tout type de projet Spring Boot (web, batch, etc.)
- Ordonner avec @Order

Utilisation de CommandLineRunner

- Charger des données
- Exécuter un batch
- Définir des propriétés d'environnement
- Avant la fin de l'exécution de la méthode main()

```
1 package com.example.demo;
2
3+ import org.springframework.beans.factory.annotation.Autowired;[]
4
5 @Component
6 public class Console implements CommandLineRunner {
7
8     @Autowired
9     Service service;
10
11     @Override
12     public void run(String... args) throws Exception {
13         service.tracer("Bonjour ");
14     }
15
16 }
```

Conclusion

Conclusion

- Rappels Spring
- Démarrer une application Spring Boot
- Autoconfiguration et starters
- Test d'intégration avec Spring Boot
- Monitoring avec Actuator
- Interface CommandLineRunner