

Introduction à SpringBoot

Présentation de la philosophie et mise en
oeuvre





Pourquoi SpringBoot?

Historique

- ❑ Les Standards **JEE** et leurs implémentations ont un **cycle d'évolution long**
- ❑ Les applications **JEE** (JEE<7) étaient **lourdes, longues à démarrer** et à **configurer** (beaucoup, beaucoup de configuration)
- ❑ **Spring** Framework propose une alternative, des serveurs **plus légers** (uniquement un container de servlet nécessaire) et un **peu moins de configuration** (.xml)
- ❑ JEE 7 Proche de la philosophie de Spring
- ❑ **Spring** Framework **ne suit pas le standard JEE** et est plus réactif sur des évolutions mais fournit pas de portabilité et de standardisation

Bilan

JEE



Old, standard, robust, innovation slow but compliant

Spring



None Standard, Quick innovation, Quick interconnection with new tools, less guarantees

Différence entre Spring et Springboot ?



VS



Différence entre Spring et Springboot ?



Spring

- Beaucoup de modules (ajout à l'aide d'injection de dépendance)
- Basé sur **l'injection de dépendances**
- Enormément de chose customisable
- Beaucoup de configuration
- pas un standard JEE



Springboot

- basé sur un Spring « déjà » configuré
- Ajout de « Starter », modules déjà configurés (gestion de dépendances)
- Simplification (extrême) de la configuration
- Grande utilisation des **annotations**
- Convention Over Configuration**
- Moins Customizable (facilement)
- Server Embarqué (e.g Apache Tomcat)

<https://www.baeldung.com/spring-vs-spring-boot>

Différence entre Spring et Springboot ?



Spring



Springboot

Spring Framework	Spring Boot Framework
The primary feature is Dependency Injection	Autoconfiguration is the primary feature of Spring Boot
we need to set up the server explicitly for the testing procedure.	offers embedded server such as Jetty and Tomcat, etc
For smaller tasks, developers need to write a boilerplate code	Reduction in boilerplate code
Does not provide an in-memory Database	Provide several plugins to work with embedded servers and some in-memory databases such as H2
Developers have to define dependencies manually in the pom.xml file	Starter concept in pom.xml file internally handles the required dependencies

<http://www.fusion-reactor.com/blog/the-difference-between-spring-framework-vs-spring-boot/>

Configuration pour la création JSP Web App



Spring



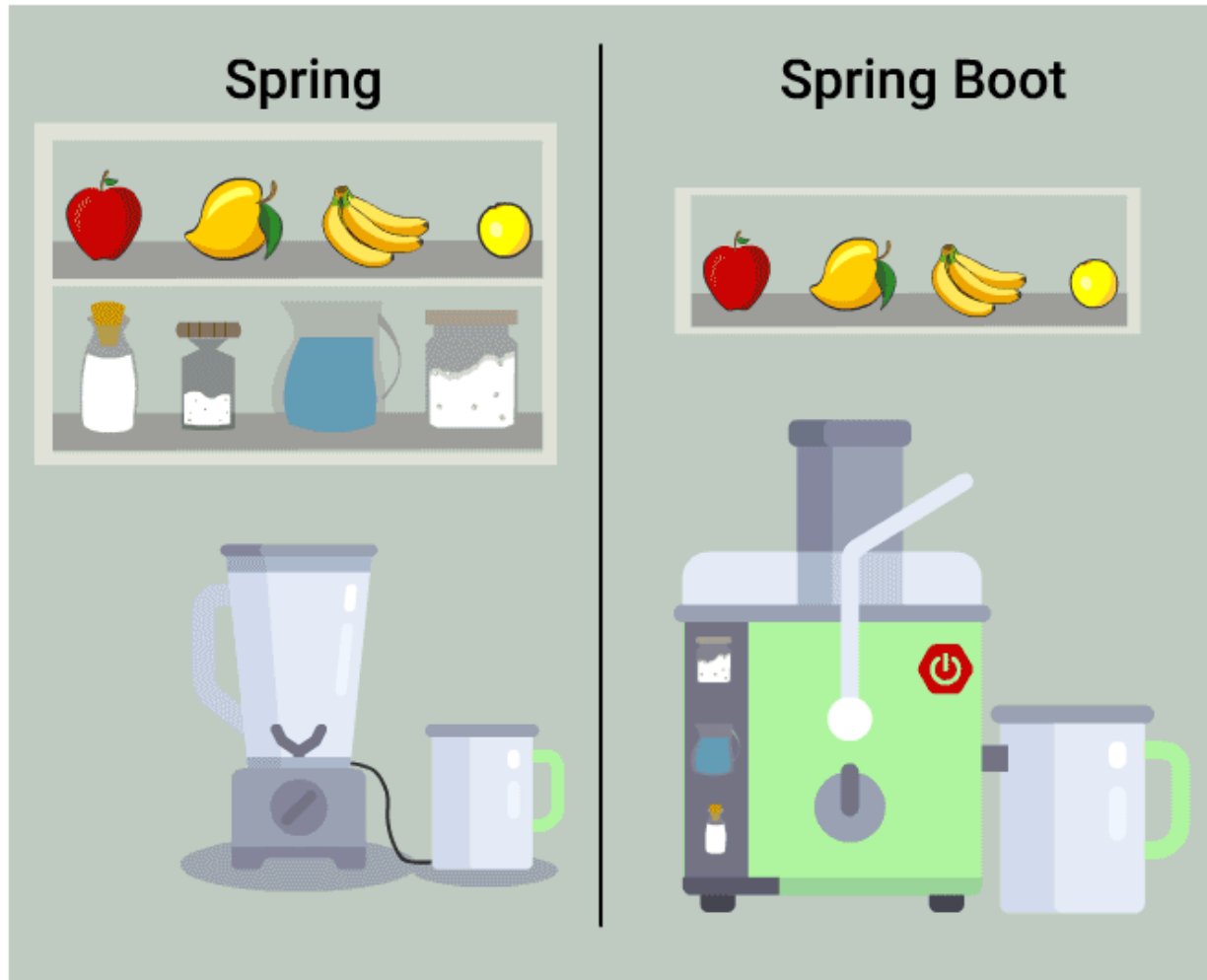
Springboot

```
public class MyWebAppInitializer implements WebApplicationInitializer {  
  
    @Override  
    public void onStartUp(ServletContext container) {  
        AnnotationConfigWebApplicationContext context  
            = new AnnotationConfigWebApplicationContext();  
        context.setConfigLocation("com.baeldung");  
  
        container.addListener(new ContextLoaderListener(context));  
  
        ServletRegistration.Dynamic dispatcher = container  
            .addServlet("dispatcher", new DispatcherServlet(context));  
  
        dispatcher.setLoadOnStartup(1);  
        dispatcher.addMapping("/");  
    }  
}
```

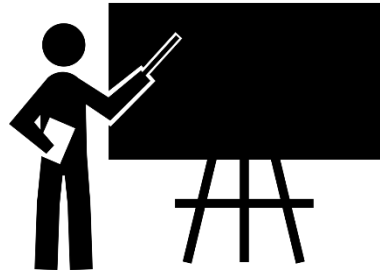
```
spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp
```

```
@EnableWebMvc  
@Configuration  
public class ClientWebConfig implements WebMvcConfigurer {  
    @Bean  
    public ViewResolver viewResolver() {  
        InternalResourceViewResolver bean  
            = new InternalResourceViewResolver();  
        bean.setViewClass(JstlView.class);  
        bean.setPrefix("/WEB-INF/view/");  
        bean.setSuffix(".jsp");  
        return bean;  
    }  
}
```


Bilan



<http://www.fusion-reactor.com/blog/the-difference-between-spring-framework-vs-spring-boot/>



Rappel de concepts

Parce que c'est mieux de savoir comment
ça marche!

Les outils de SpringBoot (et de Spring)

❑ Injection de dépendance

→ Permet de découpler les dépendances entre les objets et de réaliser de l'inversion de contrôle

❑ Annotations

→ Ajout de configuration dans le code (e.g moins de fichier .xml),

→ Ajout de comportements spécifiques compilation/runtime

❑ Outils de Gestion de dépendance (e.g Maven)

→ Permet de résoudre les dépendances extérieures nécessaires au fonctionnement de l'application

→ Certains outils permettent également de gérer le cycle de vie de l'application





Injection de dépendances

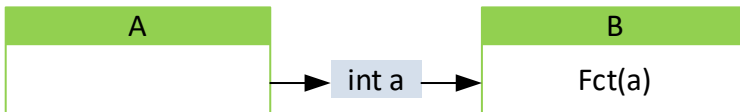
Le problème du couplage fort

□ La notion de couplage selon Pressman

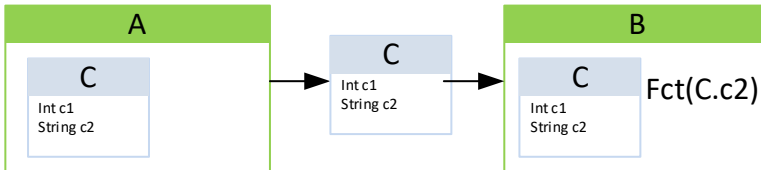
Sans Couplage



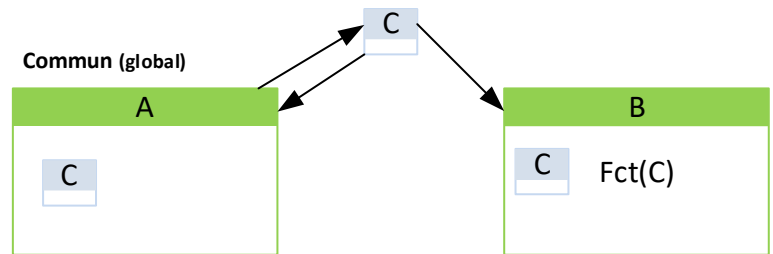
Par données



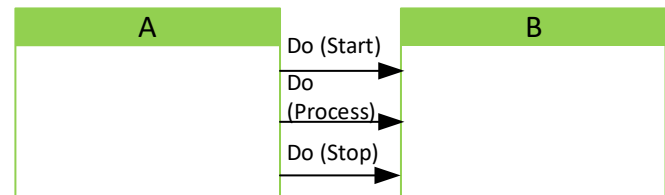
Par paquet



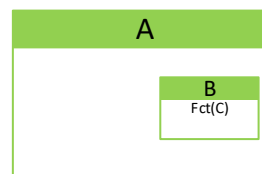
Externe



Par Control



Par contenu



Le problème du couplage fort

- ❑ Sur des gros projets aboutit à l'**anti-pattern spaghetti**
- ❑ Les composants sont **difficilement réutilisables**
- ❑ Les composants sont **difficilement testables**
- ❑ Si des ressources communes sont utilisées des **inter blocages** peuvent survenir

→ Besoin de **réduire le couplage** entre les objets autant que possible



[https://fr.wikipedia.org/wiki/Couplage_\(informatique\)](https://fr.wikipedia.org/wiki/Couplage_(informatique))

Injection de dépendances

□ Inverse of Control (IoC)

▪ Définition

Le contrôle et le cycle de vie d'objets ou de portions du programme est transféré à un tiers (container, Framework)

▪ Plusieurs mises en œuvre

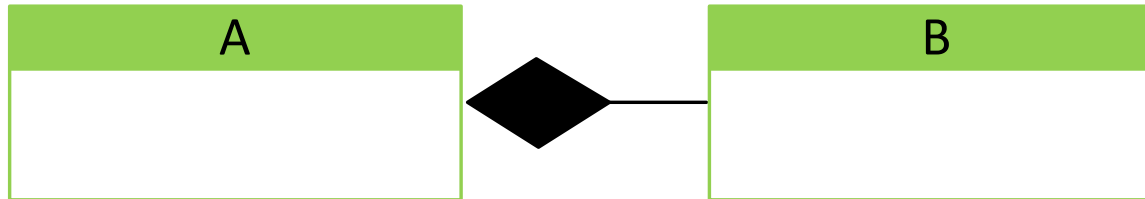
- Strategy design pattern,
- Service Locator pattern,
- Factory pattern,
- Dependency Injection (DI).



Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

Injection de dépendance

- Inverse of Control (IoC)
 - Approche Naive



```
package com.naive;

public class A {
    private B b;

    public A() {
        b=new B();
    }

    public String getData(){
        return b.getData();
    }
}
```

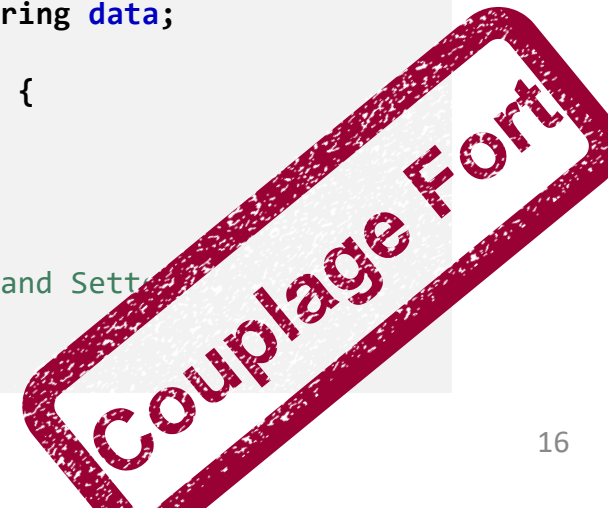
```
package com.naive;

public class B {
    private String data;

    public B() {
    }

    // Getter and Setter

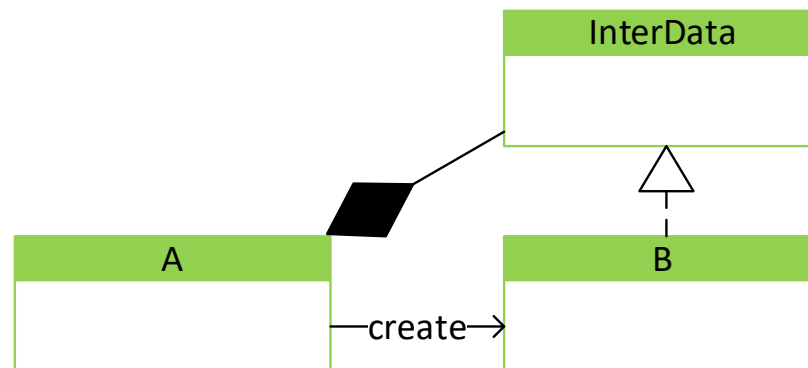
}
```



Injection de dépendance

- Inverse of Control (IoC)
 - Utilisation d'interface

```
public interface InterData {  
    public String getAllData();  
}
```



```
public class A {  
    private InterData b;  
  
    public A() {  
        b=new B();  
    }  
  
    public String getData(){  
        return b.getAllData();  
    }  
}
```

```
public class B implements InterData {  
    private String data;  
  
    public B() {  
    }  
    @Override  
    public String getAllData() {  
        return getData();  
    }  
    // Getter and setter  
}
```

Couplage plus faible

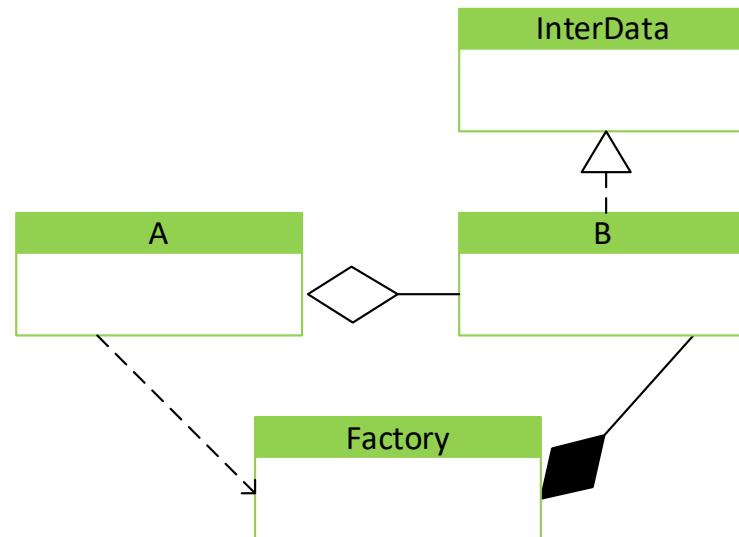
Injection de dépendance

- Inverse of Control (IoC)
 - Utilisation d'une Fabrique

```
public interface InterData {  
    // Idem  
}
```

```
public class B implements InterData {  
    // Idem  
}
```

```
public class A {  
    private InterData b;  
  
    public A() {  
        b= Factory.getInterDataInstance();  
    }  
    public String getData(){  
        return b.getAllData();  
    }  
}
```



```
public class Factory {  
    private static InterData instance;  
    public static void  
        setInstance(InterData processor) {  
        instance = processor;  
    }  
    public static InterData  
        getInterDataInstance() {  
        if (instance == null) {  
            InterData b = new B();  
            setInstance(b);  
            return b;  
        }  
        return instance;  
    }  
}
```

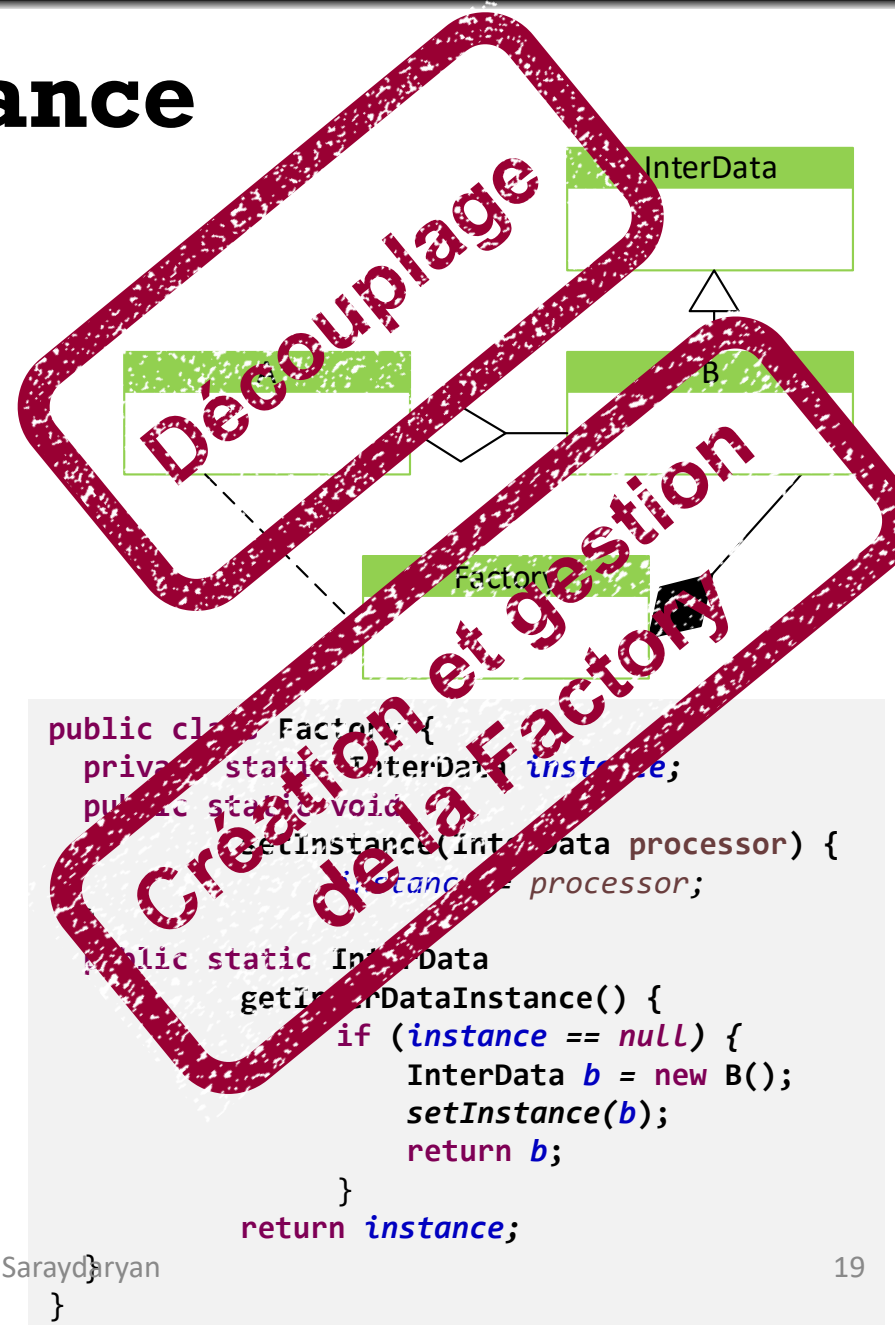
Injection de dépendance

- Inverse of Control (IoC)
 - Utilisation d'une Fabrique

```
public interface InterData {  
    // Idem  
}
```

```
public class B implements InterData {  
    // Idem  
}
```

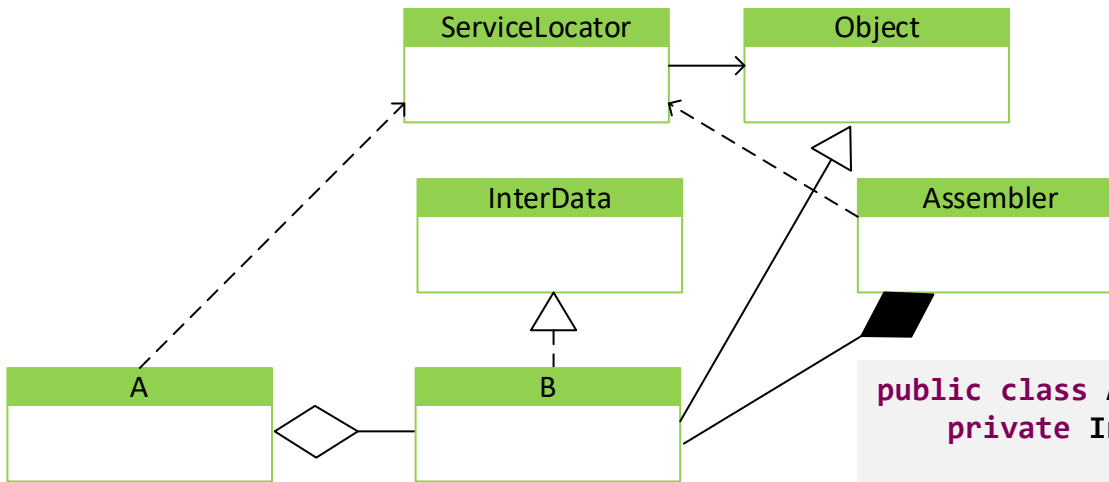
```
public class A {  
    private InterData b;  
  
    public A() {  
        b= Factory.getInterDataInstance();  
    }  
    public String getData(){  
        return b.getAllData();  
    }  
}
```



```
public class Factory {  
    private static InterData instance;  
    public static void  
        getInstance(int data processor) {  
        instance = processor;  
    }  
    public static InterData  
        getInstance() {  
        if (instance == null) {  
            InterData b = new B();  
            setInstance(b);  
            return b;  
        }  
        return instance;  
    }  
}
```

Injection de dépendance

- Inverse of Control (IoC)
 - Service Locator



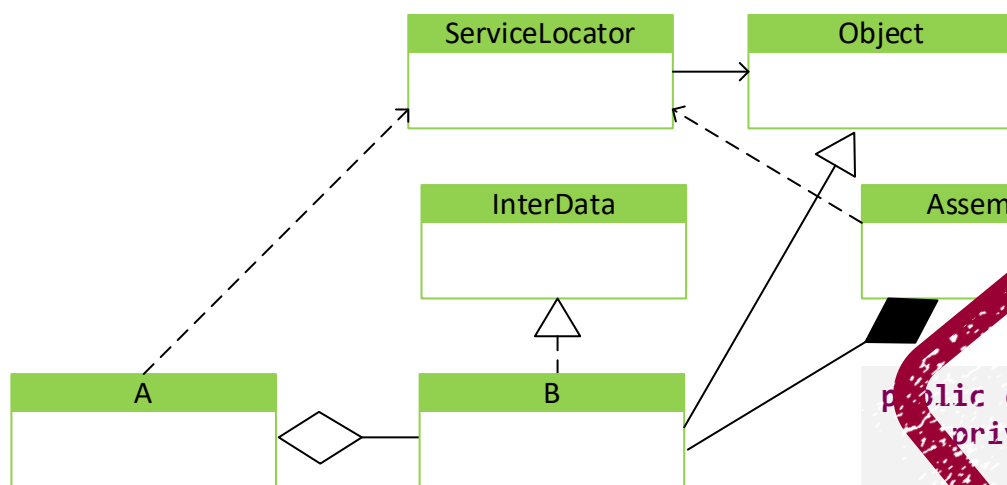
```
public class A {
    private InterData b;

    public A() {
        b =
        (InterData)ServiceLocator.getService("inter_data");
    }

    public String getData(){
        return b.getAllData();
    }
}
```

Injection de dépendance

- Inverse of Control (IoC)
 - Service Locator



**Appel explicite
Du service recherché**

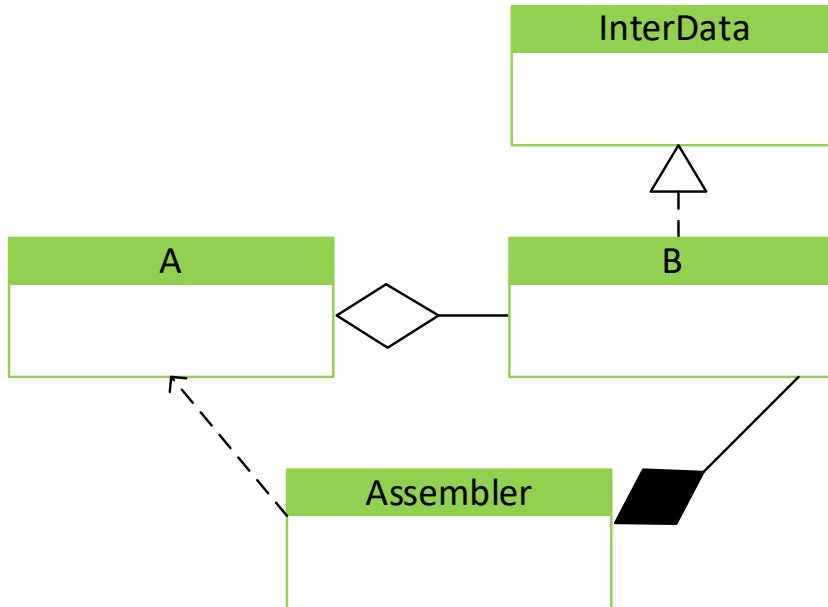
```
public class A {
    private InterData b;

    public A() {
        b =
        (InterData) ServiceLocator.getService("inter_data");
    }

    public String getData(){
        return b.getAllData();
    }
}
```

Injection de dépendance

- Inverse of Control (IoC)
 - Injection de dépendances



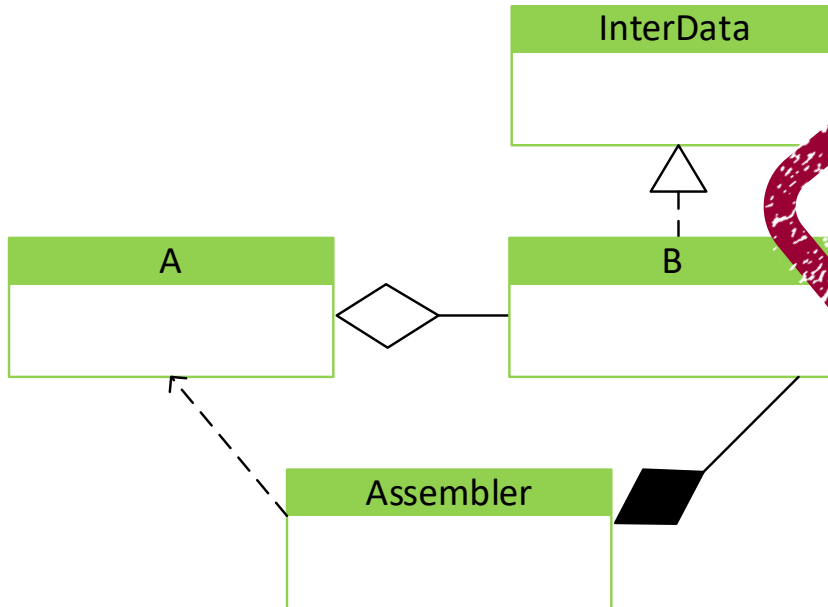
```
public class A {
    private InterData b;

    public A() {
    }

    public String getData(){
        return b.getAllData();
    }
}
```

Injection de dépendance

- Inverse of Control (IoC)
 - Injection de dépendances



Injection du service
automatique

```
public class A {
    private InterData b;

    public A() {
    }

    public String getData() {
        return b.getAllData();
    }
}
```

Inversion de Contrôle

Injection de dépendance

- ❑ Injection de dépendances (DI)
 - **Injection via les mutateurs**
 - **Injection via les constructeur**

- ❑ Usage dans Spring
 - Les implémentations à injecter doivent être des Beans contenant des mutateurs pour l'injection
 - **ApplicationContext** de Spring permet de tout mettre en place
 - Utilise les fichiers de configuration XML (e.g Déclaration des beans)
 - Création des instances à la demande
 - Possibilité d'utiliser les **annotations** afin d'éviter les fichiers XML (e.g **@Autowire**)



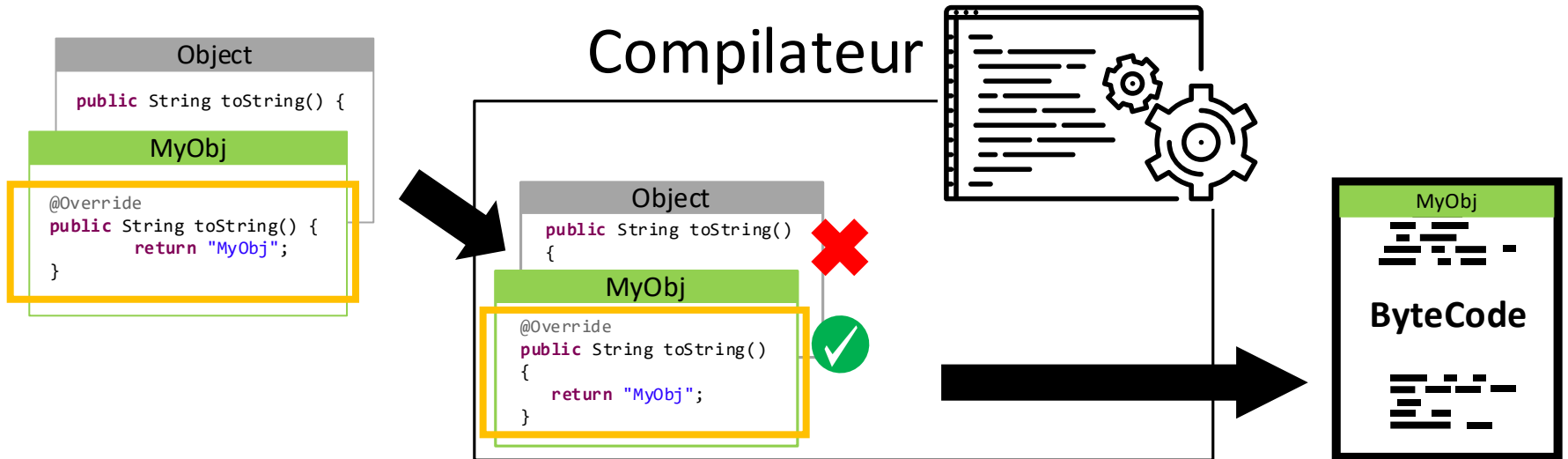
Annotations

Qu'est ce qu'une annotation Java

- ❑ Informations supplémentaires (métadonnées) ajoutées dans le code source pouvant changer le comportement d'exécution ou de compilation de ce dernier (apparu dans Java 1.5)
- ❑ Peut être utilisé/gardé (usage de l'outil Annotation Processing Tool)
 - À la compilation (e.g @Override)
 - Au runtime (e.g @Resource)
- ❑ Présence d'un set d'annotations standards (@Override, @Deprecated)
- ❑ Customisation des annotations possibles pour différents usages
 - Documentation
 - Génération de code
 - Vérification
 - Configuration
 - ...

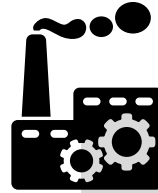
Exemple d'annotations

- ❑ @Override : surcharge d'une méthode héritée



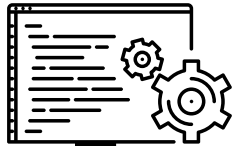
Exemple d'annotations

Runtime

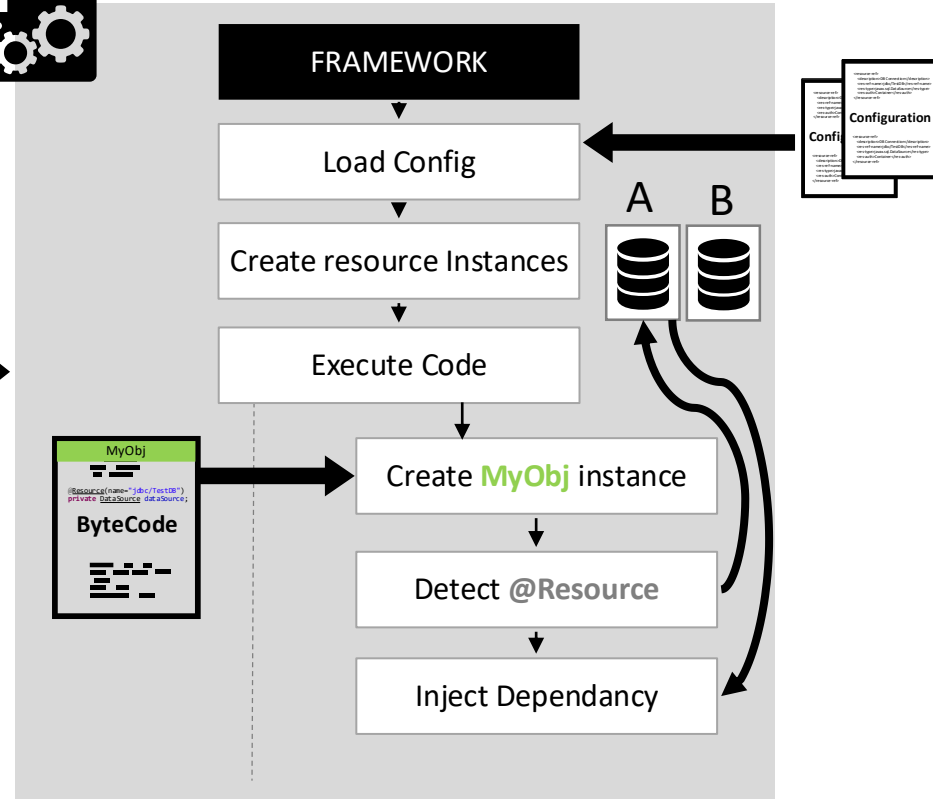
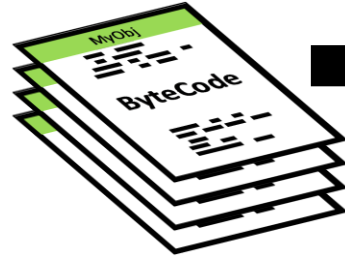


```
MyObj  
@Resource(name="jdbc/TestDB")  
private DataSource dataSource;
```

Compilateur



```
MyObj  
@Resource(name="jdbc/TestDB")  
private DataSource dataSource;  
  
ByteCode
```





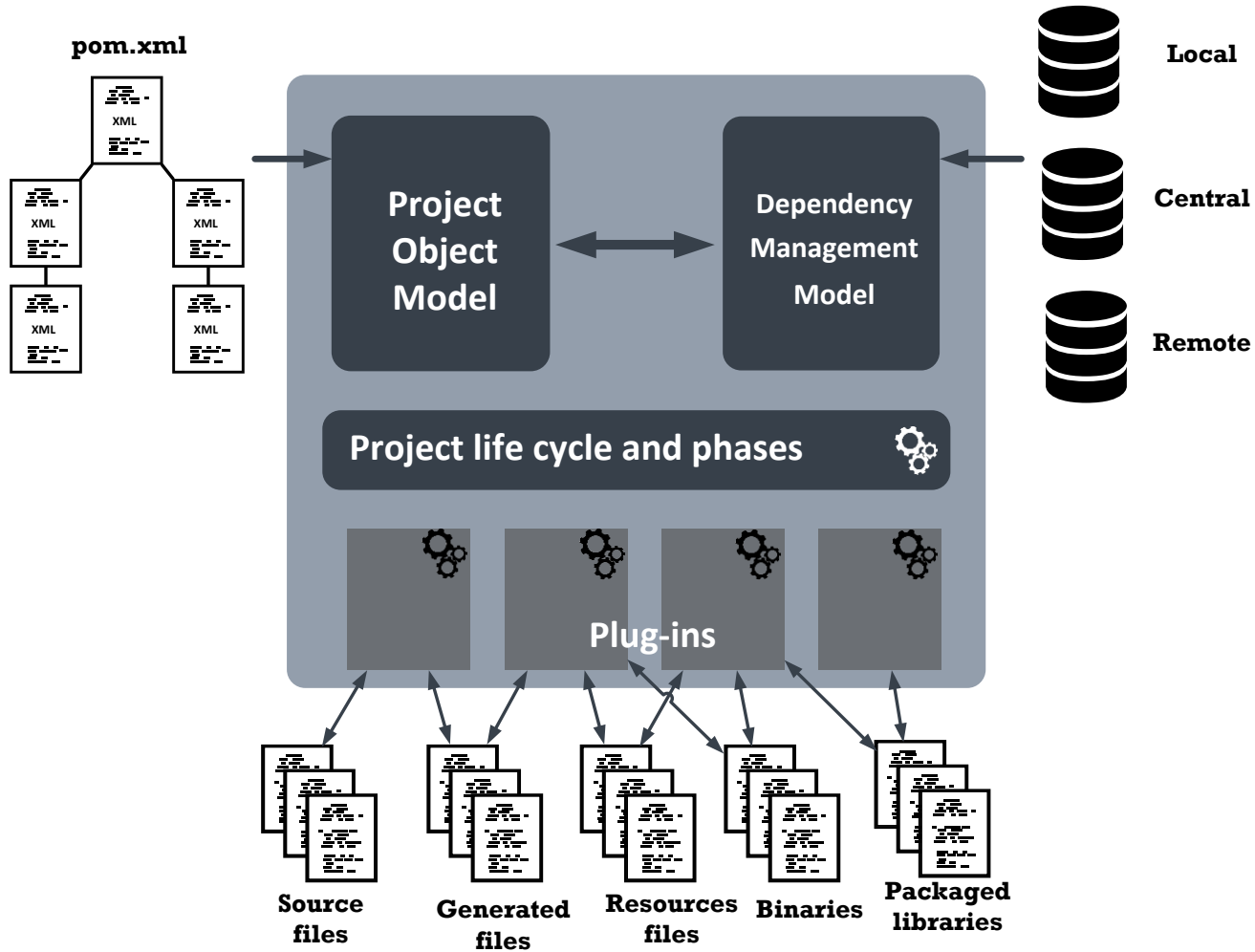
Gestionnaire de dépendances

Gestionnaire de dépendances MAVEN

- Comment avoir un cadre de travail homogène pour mes équipes ?
- Comment gérer de multiples dépendances de façon uniforme?
- Comment automatiser le test , la compilation et le packaging ?
- Comment Post traiter le résultat de mon développement ?



Maven

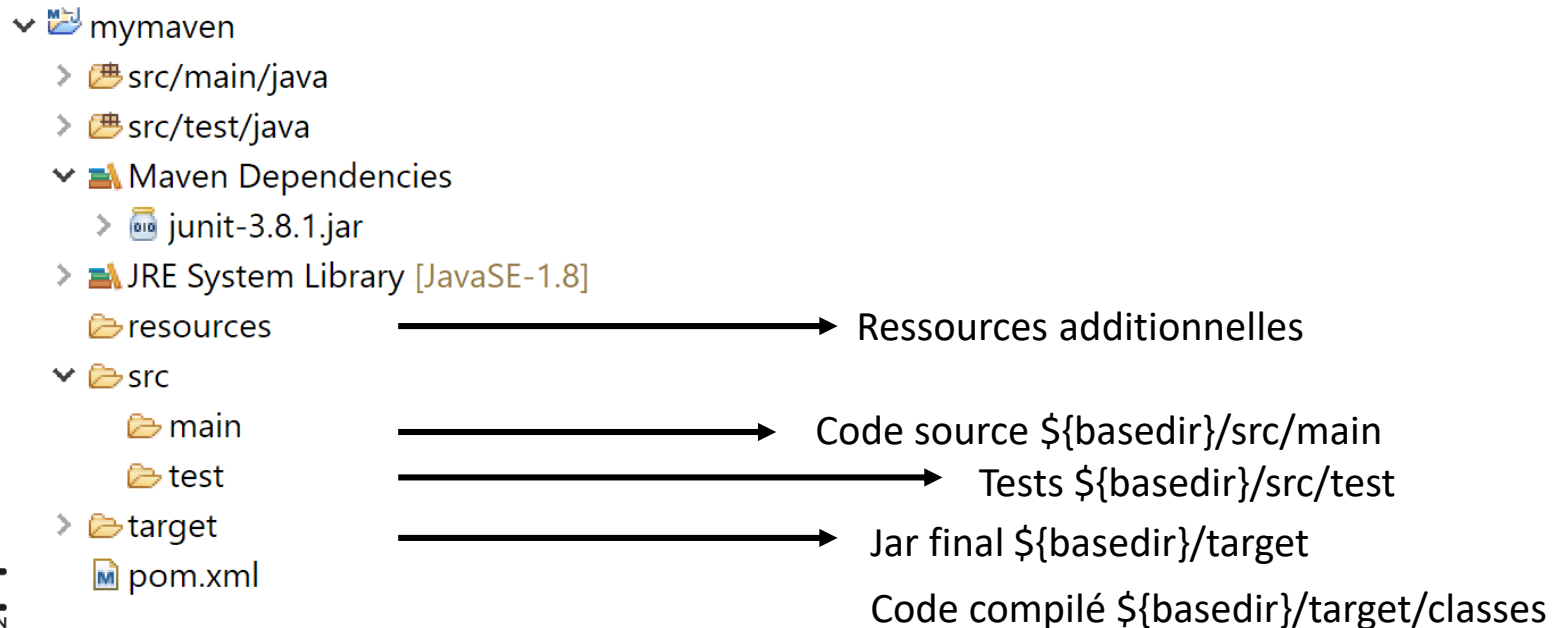


Convention: une même organisation

- ❑ Création d'un projet maven
 - Usage du plugin **archetype** pour créer la structure du projet

```
mvn --version
```

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app \  
-DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 \  
-DinteractiveMode=false
```



Cycle de vie (build)

Validate

Vérifie que le projet est valide et que toutes les informations nécessaires sont accessibles

Compile

Compile le code source du projet

Test

Test le code compilé à l'aide du Framework de tests unitaires approprié (not required)

Package

Package le code compilé dans un format distribuable (e.g jar, war)

Verify

Effectue des tests d'intégrations

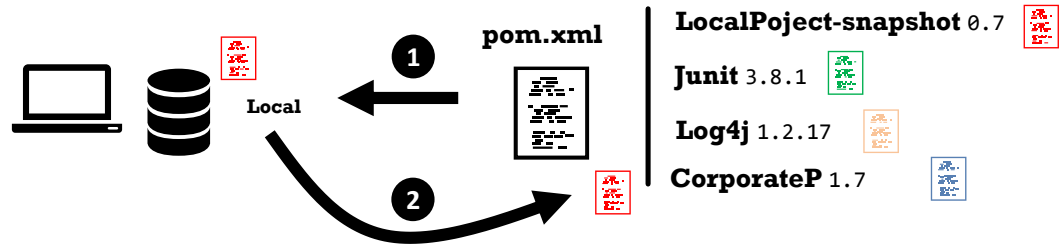
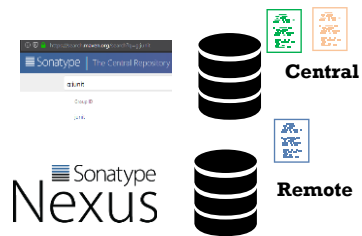
Install

Install le package dans le local repository pour être utilisé en tant que dépendance par d'autres projets

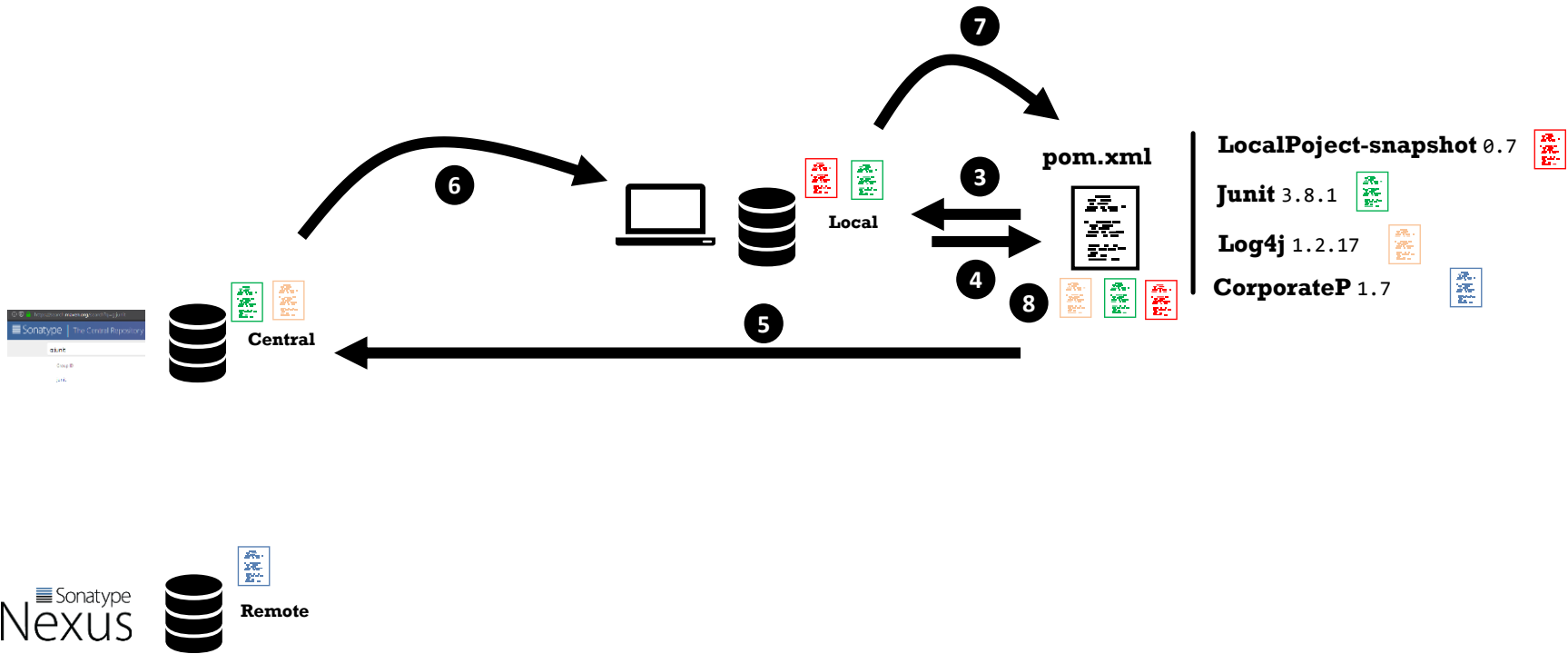
Deploy

Effectué sur l'environnement de build, copie le package final sur un répertoire distant pour le partager avec d'autres développeurs

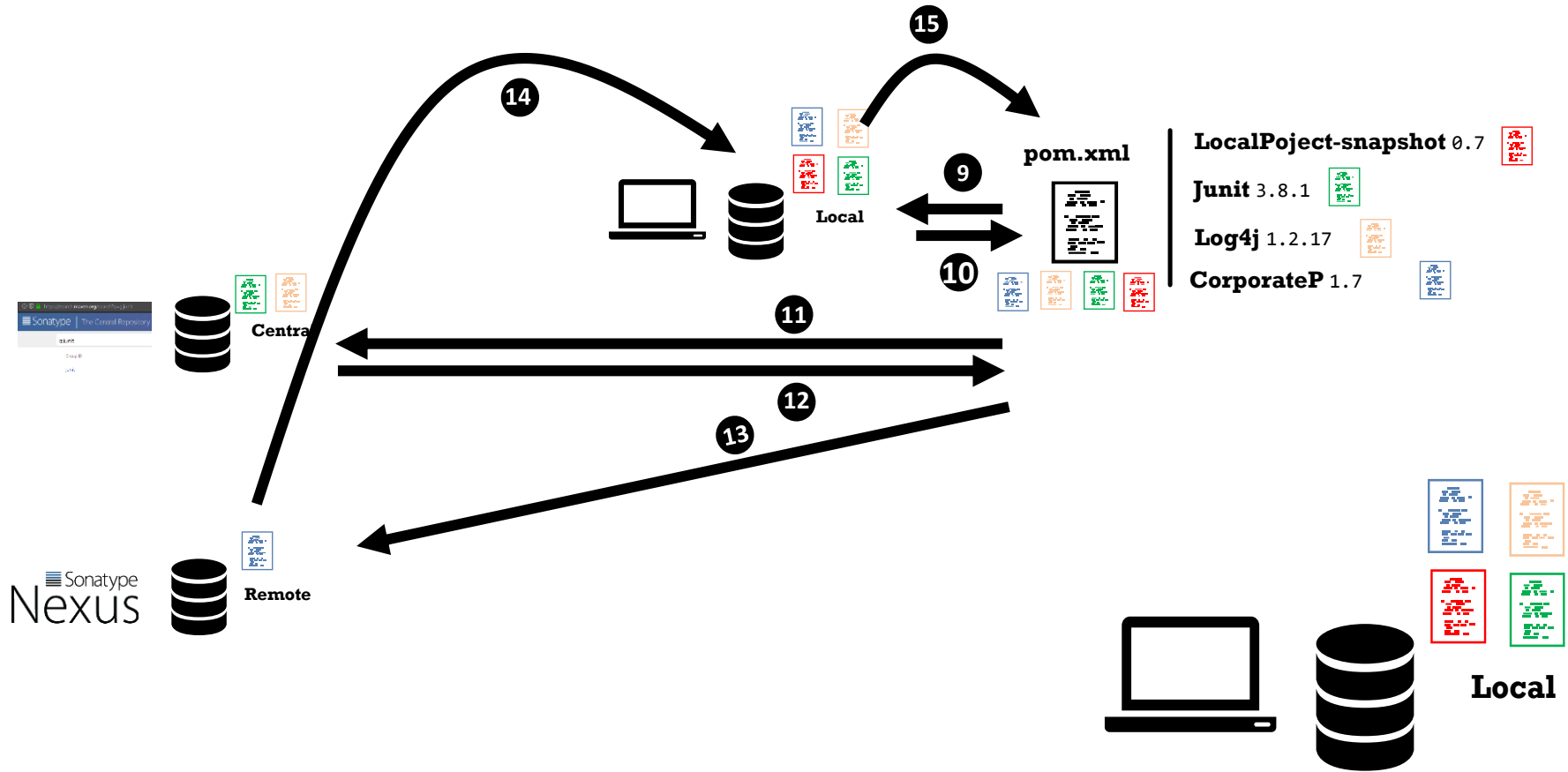
Gestion des dépendances



Gestion des dépendances



Gestion des dépendances



Description : Pom.xml

https://maven.apache.org/pom.html#What_is_the_POM

```
<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.course</groupId>
<artifactId>my.maven</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

```
<packaging>jar</packaging>
```

```
<name>my.maven</name>
<url>http://maven.apache.org</url>
```

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    <type>jar</type>
    <optional>true</optional>
  </dependency>
</dependencies>
```

Description : Pom.xml

https://maven.apache.org/pom.html#What_is_the_POM

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.0.1</version>
      <executions>
        <execution>
          <id>execution1</id>
          <phase>package</phase>
          <configuration>
            <show>private</show>
            <nohelp>>true</nohelp>
          </configuration>
          <goals>
            <goal>javadoc</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.0</version>
      <configuration>
        <verbose>>true</verbose>
        <executable>${basedir}/target
      </executable>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```



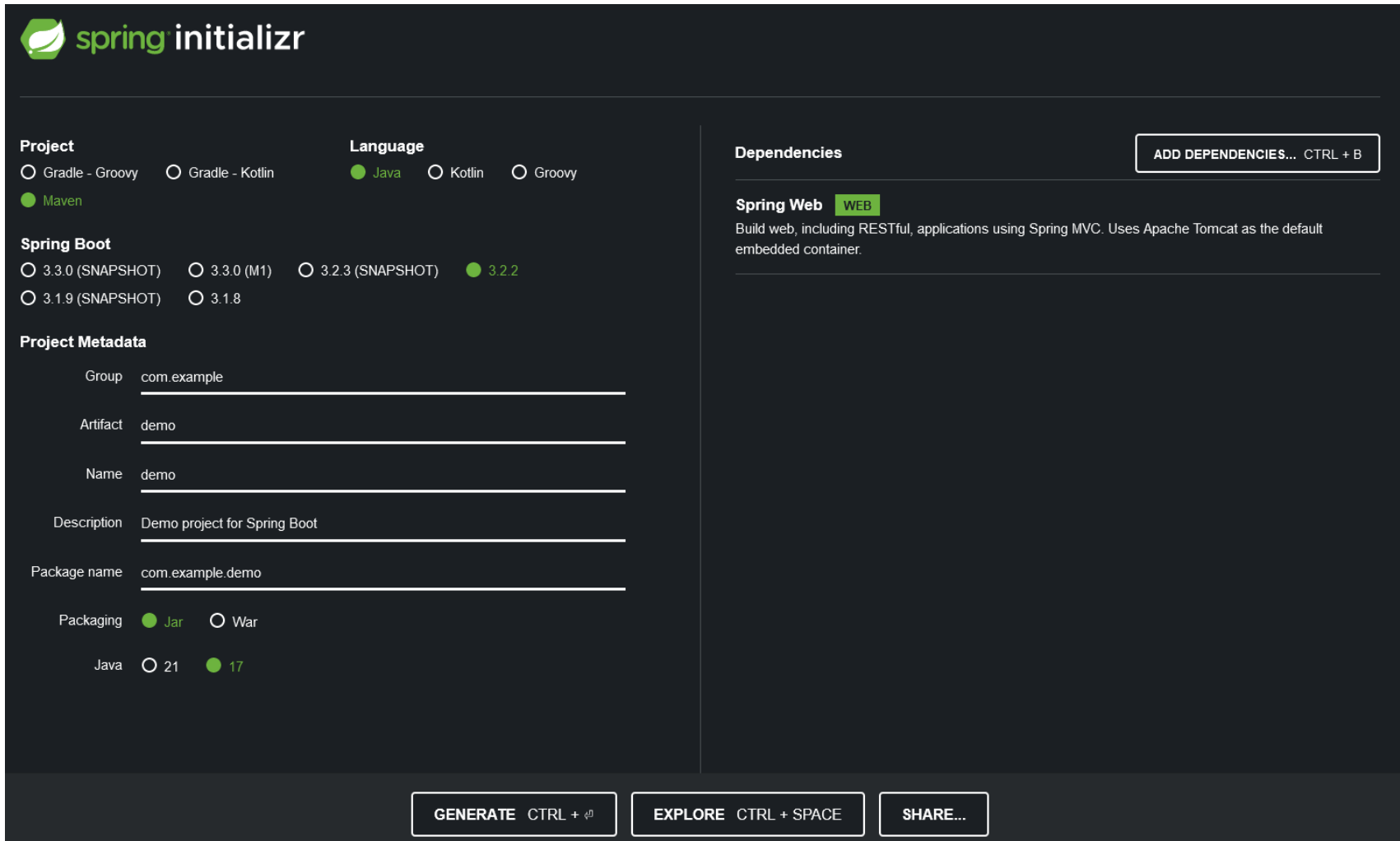
SpringBoot Getting started !

Principales propriétés de Springboot

- ❑ Autoconfiguration
- ❑ Starter
- ❑ Injection de dépendances (hérité de Spring)
- ❑ Principaux modules:
 - *spring-boot-starter-data-jpa*
 - *spring-boot-starter-security*
 - *spring-boot-starter-test*
 - *spring-boot-starter-web*
 - *spring-boot-starter-thymeleaf*



Création d'un projet - <https://start.spring.io/>

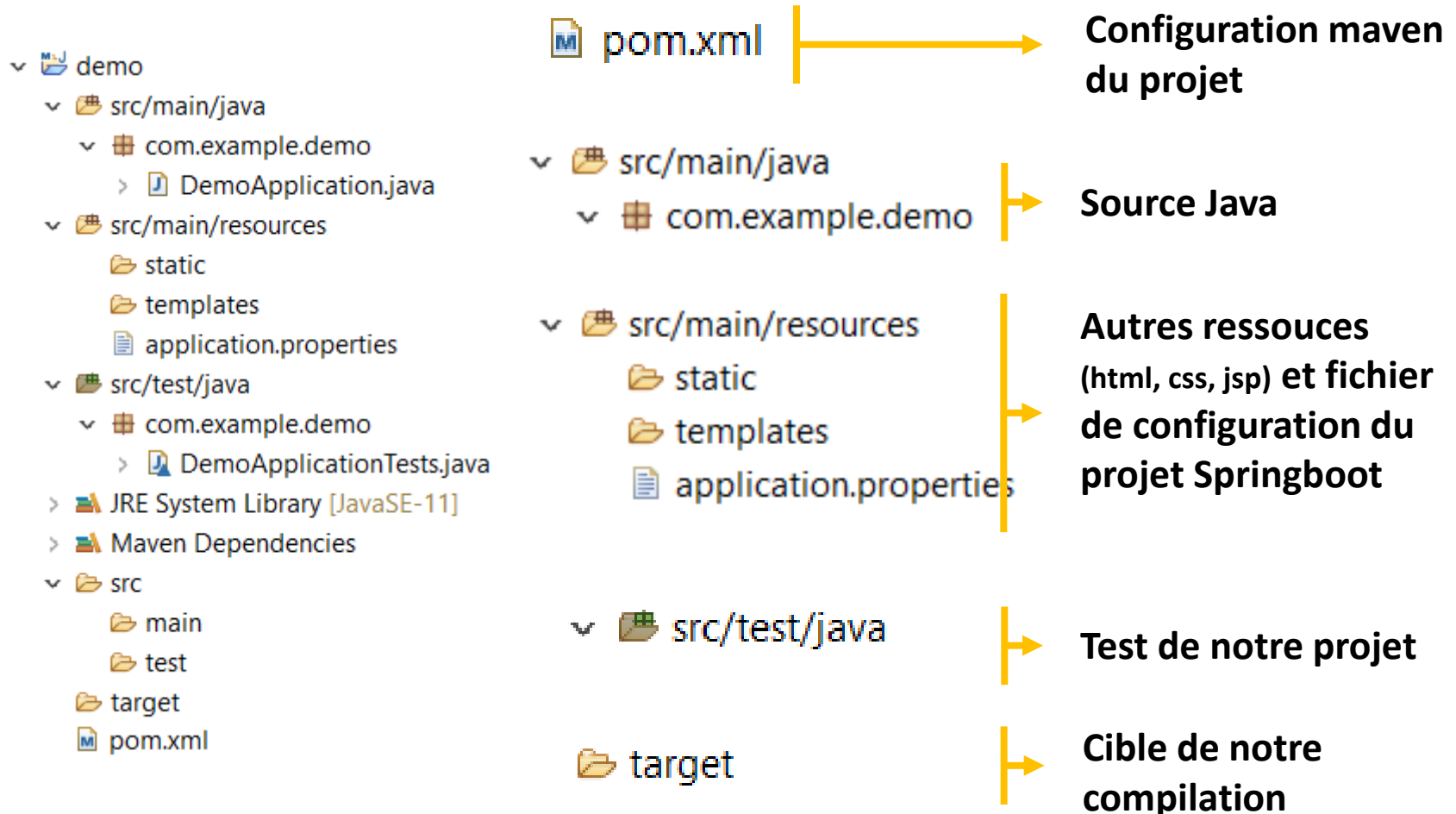


The screenshot shows the Spring Initializr web application interface. The top left features the Spring logo and the text "spring initializr". The main content area is divided into several sections:

- Project:** Radio buttons for "Gradle - Groovy", "Gradle - Kotlin", "Maven" (selected), "Java" (selected), "Kotlin", and "Groovy".
- Language:** Radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Radio buttons for "3.3.0 (SNAPSHOT)", "3.3.0 (M1)", "3.2.3 (SNAPSHOT)", "3.2.2" (selected), "3.1.9 (SNAPSHOT)", and "3.1.8".
- Project Metadata:** Text input fields for "Group" (com.example), "Artifact" (demo), "Name" (demo), "Description" (Demo project for Spring Boot), and "Package name" (com.example.demo).
- Packaging:** Radio buttons for "Jar" (selected) and "War".
- Java:** Radio buttons for "21" and "17" (selected).
- Dependencies:** A section titled "Spring Web" with a "WEB" tag and a description: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container." A button "ADD DEPENDENCIES... CTRL + B" is located at the top right of this section.

At the bottom of the form, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE..."

Création d'un projet - <https://start.spring.io/>



À vous de Jouer !

- Créer une application SpringBoot depuis <https://start.spring.io/>
 - Ajouter la dépendance suivante:
 - Spring Web
- Importer votre projet dans eclipse
- Analyser l'arborescence de votre projet
- Lancer votre application



Création d'un RestController

☐ Requêtes simples

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloRestCrt {

    @RequestMapping("/hello1")
    public String sayHello() {
        return "Hello 1 !!!!!";
    }

    @GetMapping("/hello2")
    public String sayHello2() {
        return "Hello 2 !!!!!";
    }

    @RequestMapping(method = RequestMethod.GET, value = "/hello3")
    public String sayHello3() {
        return "Hello 3 !!!!!";
    }
}
```

@RestController
Annotation permettant de déclencher des comportements liés à des requêtes HTTP

@RequestMapping
Méthodes Java déclenchée par une requête HTTP GET sur /hello1, /hello2, /hello3

Création d'un RestController

☐ Requêtes avancées

```
@RestController
public class HeroAdvRestCrt {

    @RequestMapping(method=RequestMethod.POST, value="/addhero")
    public void addHero(@RequestBody Hero hero) {
        System.out.println(hero);
    }

    @RequestMapping(method=RequestMethod.GET, value="/msg/{id1}/{id2}")
    public String getMsg(@PathVariable String id1, @PathVariable String id2)
    {
        String msg1=id1;
        String msg2=id2;
        return "Composed Message: msg1:"+msg1+"msg2:"+msg2;
    }

    @RequestMapping(method=RequestMethod.GET, value="/parameters")
    public String getInfoParam(@RequestParam String param1,@RequestParam
    String param2) {
        return "Parameters: param1:"+param1+"param2:"+param2;
    }
}
```

@RequestBody

Récupère le body de la requête HTTP et tente de la convertir en objet Hero

@PathVariable

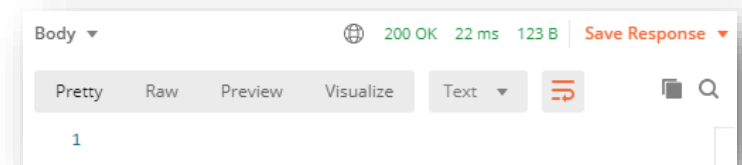
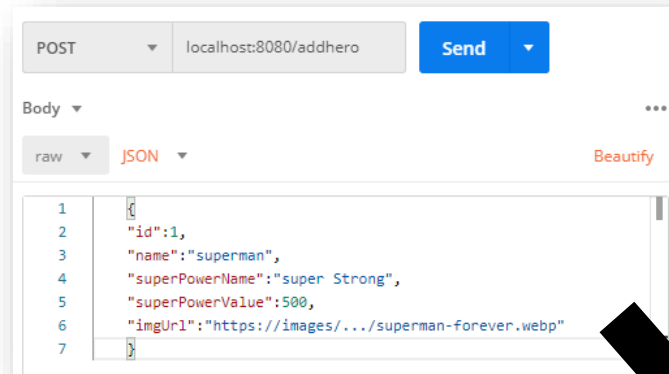
Récupère des variables dans l'URL

@RequestParam

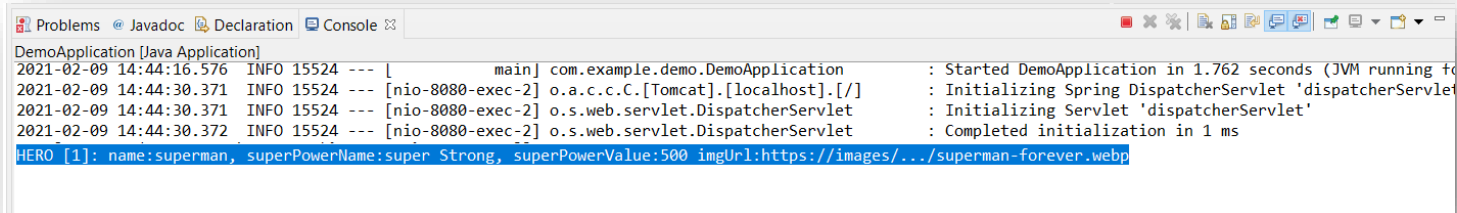
Récupère un paramètre de la requête

Création d'un RestController

❑ Requêtes avancées

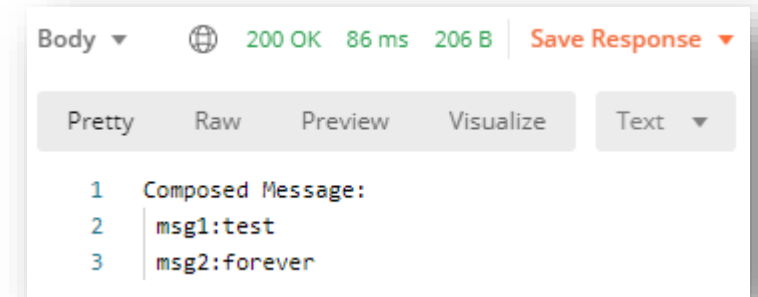
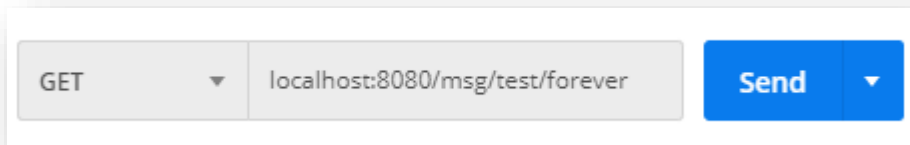


```
@RequestMapping(method=RequestMethod.POST,value="/addhero")
public void addHero(@RequestBody Hero hero) {
    System.out.println(hero);
}
...
```



Création d'un RestController

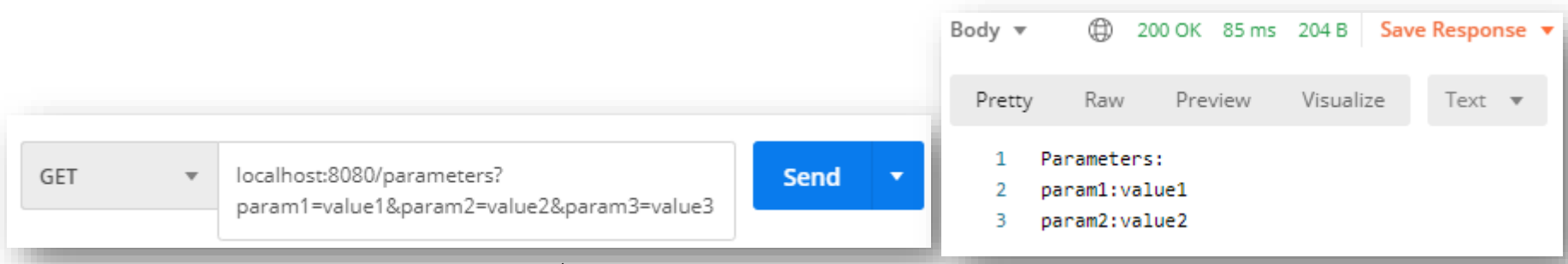
❑ Requêtes avancées



```
@RequestMapping(method=RequestMethod.GET, value="/msg/{id1}/{id2}")
public String getMsg(@PathVariable String id1, @PathVariable
String id2) {
    String msg1=id1;
    String msg2=id2;
    return "Composed Message: msg1:"+msg1+"msg2:"+msg2;
}
...
```

Création d'un RestController

❑ Requêtes avancées



The screenshot shows a REST client interface. On the left, a dropdown menu is set to 'GET'. The URL field contains 'localhost:8080/parameters?param1=value1¶m2=value2¶m3=value3'. A blue 'Send' button is to the right. On the right, the response is displayed in a 'Body' panel. The status is '200 OK', the response time is '85 ms', and the size is '204 B'. The response body is shown in 'Pretty' format as a list of three items: '1 Parameters:', '2 param1:value1', and '3 param2:value2'. Two large black arrows point from the code block below to the 'Send' button and the response panel.

```
@RequestMapping(method=RequestMethod.GET,value="/parameters")  
  
public String getInfoParam(@RequestParam String param1,@RequestParam String param2) {  
    return "Parameters: param1:"+param1+"param2:"+param2;  
}  
...
```


À vous de Jouer !

- ❑ Ajouter 2 RestControllers à votre projet
 - MovieRestCrt
 - ActorRestCrt
- ❑ Ajouter les modèles suivants:
 - **Actor** `LastName,Surname,Birthday date (String)`
 - **Movie** `Title, Description,Date (String),Type, Budget`
- ❑ Ajouter les fonctions suivantes:
 - `/movie` , http Get, return a list of movies (à ajouter dans le constructeur)
 - `/movie/{id}`, http Post, ajoute un Movie à la liste courante
 - Faire de même pour les Actors
- ❑ Tester votre application à l'aide de POSTMAN



Création d'un Service

❑ Déclaration d'un service

```
import org.springframework.stereotype.Service;

@Service
public class HeroService {

    public String addSuffix(String msg) {
        return msg + "\n" + "Service processing";
    }
}
```

@Service

Permet la création d'un singleton de la classe courante et son injection dans d'autres classes



❑ Injection du Service

```
@RestController
public class MsgRestCrt {

    @Autowired
    private HeroService hService;

    @GetMapping("/msg")
    public String updateMsg( @RequestParam String msg) {

        return hService.addSuffix(msg);
    }
}
```

```
@RestController
public class MsgRestCrt {

    private final HeroService hService;

    MsgRestCrt(HeroService heroService) {
        this.hService=heroService;
    }

    @GetMapping("/msg")
    public String updateMsg( @RequestParam String msg) {
        return hService.addSuffix(msg);
    }
}}
```

Création d'un Service

❑ Déclaration d'un service

```
import org.springframework.stereotype.Service;

@Service
public class HeroService {

    public String addSuffix(String msg) {
        return msg + "\n" + "Service processing";
    }
}
```

@Service

Permet la création d'un singleton de la classe courante et son injection dans d'autres classes



❑ Injection du Service

```
@RestController
public class MsgRestController {

    @Autowired
    private HeroService hService;

    @GetMapping("/msg")
    public String updateMsg(@RequestParam String msg) {
        return hService.addSuffix(msg);
    }
}
```

Injection par Annotation (old way)

```
@RestController
public class MsgRestController {

    private final HeroService hService;

    MsgRestController(HeroService heroService) {
        this.hService = heroService;
    }

    @GetMapping("/msg")
    public String updateMsg(@RequestParam String msg) {
        return hService.addSuffix(msg);
    }
}
```

Injection par Constructeur

À vous de Jouer !

- Créer les services suivants qui vont contenir respectivement la liste des actors et la liste des movies (supprimer cette liste des RestCrt..)
 - ActorService
 - MovieService
- Pour chacun des services créés ajouter les fonctions:
 - Ajout / suppression / mise à jour / Récupération (add/del/update/get)
 - Ajouter les fonctions suivantes:
- Modifier vos RestCrt afin qu'ils soient compatible FULLREST et utilise les services créés.



Connexion à une base de données

- ❑ Utilisation de **JPA**

```
...  
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

Permet Ajout de la dépendance **JPA**

```
<dependency>  
<groupId>com.h2database</groupId>  
<artifactId>h2</artifactId>  
<scope>runtime</scope>  
</dependency>
```

Ajout de la dépendance **base de données H2** et **connecteur**

```
</dependencies>
```

Connexion à une base de données

❑ Création d'une **Entity**

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Hero {

    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String superPowerName;
    private int superPowerValue;
    private String imgUrl;

    public Hero() {
    }
}
```

@Entity

indique une classe persistante.
Le service JPA va ainsi créer
une table correspondante à
cette classe

@Id

Indique à JPA que l'attribut est
la clé primaire de notre table

@GeneratedValue

Indique à JPA que l'attribut
sera auto-généré

Connexion à une base de données

- ❑ Création d'un Data Access Object: **Repository**

```
import java.util.List;
import org.springframework.data.repository.CrudRepository;
import com.example.demo.model.Hero;

public interface HeroRepository extends CrudRepository<Hero, Integer> {

    public List<Hero> findByName(String name);
}
```

- Interface SpringBoot permettant la création d'**opérations CRUD** (Create Read Update Delete) d'un « *repository* » (source de données) d'un type spécifique (ici Hero)
- Utilisation de mots clé « **find** » pour créer des méthodes de recherche sur des attributs du repository visé (ici l'entité Hero)

Connexion à une base de données

- ❑ Multiples méthodes find disponibles

Table 3. Supported keywords inside method names

Keyword	Sample	JPQL snippet
Distinct	<code>findDistinctByLastnameAndFirstname</code>	<code>select distinct ... where x.lastname = ?1 and x.firstname = ?2</code>
And	<code>findByLastnameAndFirstname</code>	<code>... where x.lastname = ?1 and x.firstname = ?2</code>
Or	<code>findByLastnameOrFirstname</code>	<code>... where x.lastname = ?1 or x.firstname = ?2</code>
Is, Equals	<code>findByFirstname</code> , <code>findByFirstnameIs</code> , <code>findByFirstnameEquals</code>	<code>... where x.firstname = ?1</code>
Between	<code>findByStartDateBetween</code>	<code>... where x.startDate between ?1 and ?2</code>

- ❑ Usage de l'annotation @Query

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query("select u from User u where u.emailAddress = ?1")  
    User findByEmailAddress(String emailAddress);  
  
}
```

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.at-query>

Connexion à une base de données

- ❑ Définition de la connexion avec la base de données (si différent de H2)

Application.properties

```
server.port=8081

## FOR EXTERNAL MYSQL DB
spring.jpa.hibernate.ddl-auto = validate
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/bd
```

Connexion à une base de données

- ❑ Usage du **Repository** et de **l'Entity** depuis un **Service**

```
@Service
public class HeroService {

    @Autowired
    private HeroRepository hRepo;

    public Hero addHero(Hero h) {
        return hRepo.save(h);
    }

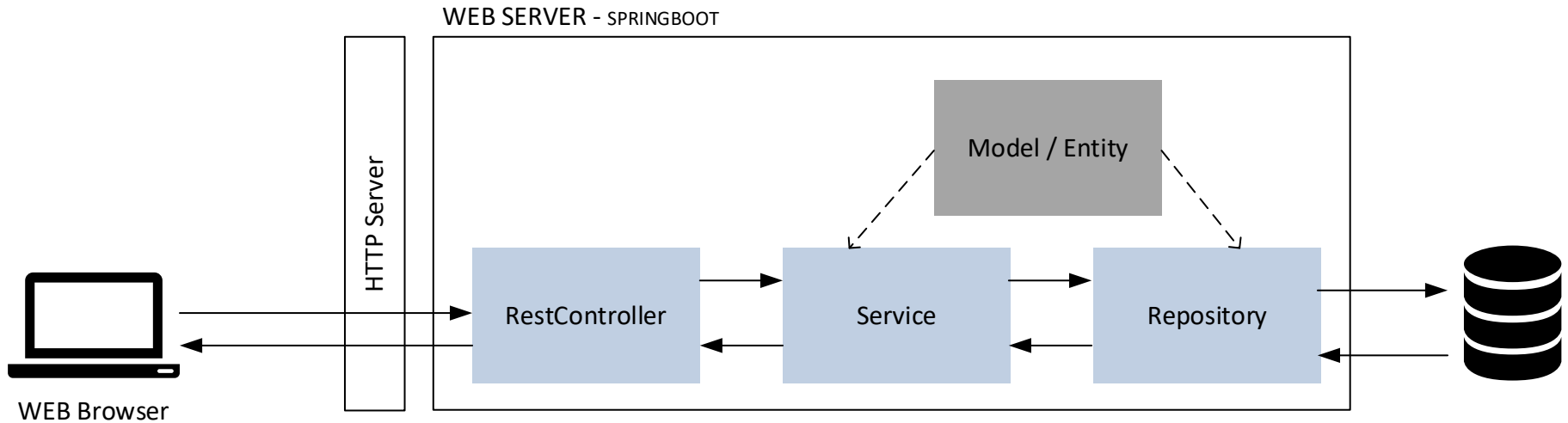
    public Optional<Hero> getHero(int id) {
        return hRepo.findById(id);
    }

    public List<Hero> findHero(String name){
        List<Hero> hList = hRepo.findByName(name);
        return hList;
    }
}
```

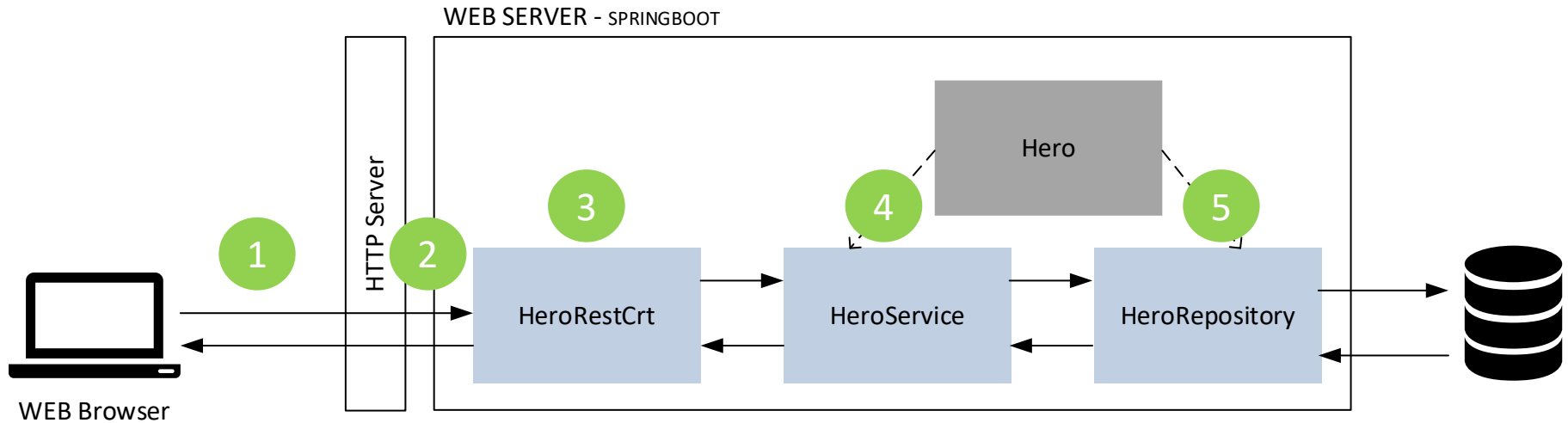
@Autowired
Injection du CRUD
Repository

Usage des méthodes
CRUD instanciées par
Springboot

Architecture Springboot Type (Rest)



Architecture Springboot Type (Rest)



- 1 HTTP GET /hero
- 2 HTTP Server transmet la requête au Rest Controller
- 3 RestController récupère les informations de la requêtes et appelle la méthode getHero de HeroService
- 4 HeroService appelle la méthode findAll() de HeroRepository
- 5 HeroRepository fait un appel à la base de données et retourne le résultat converti en objet java Hero

À vous de Jouer !

- ❑ Modifier votre application afin d'utiliser une base de donnée H2
 - Modifier vos modèles pour en faire des Entity
 - Ajouter des Repository:
 - ActorRepository
 - MovieRepository
 - Modifier vos Services pour :
 - Injecter les repository
 - Effectuer les opérations sur la base de données



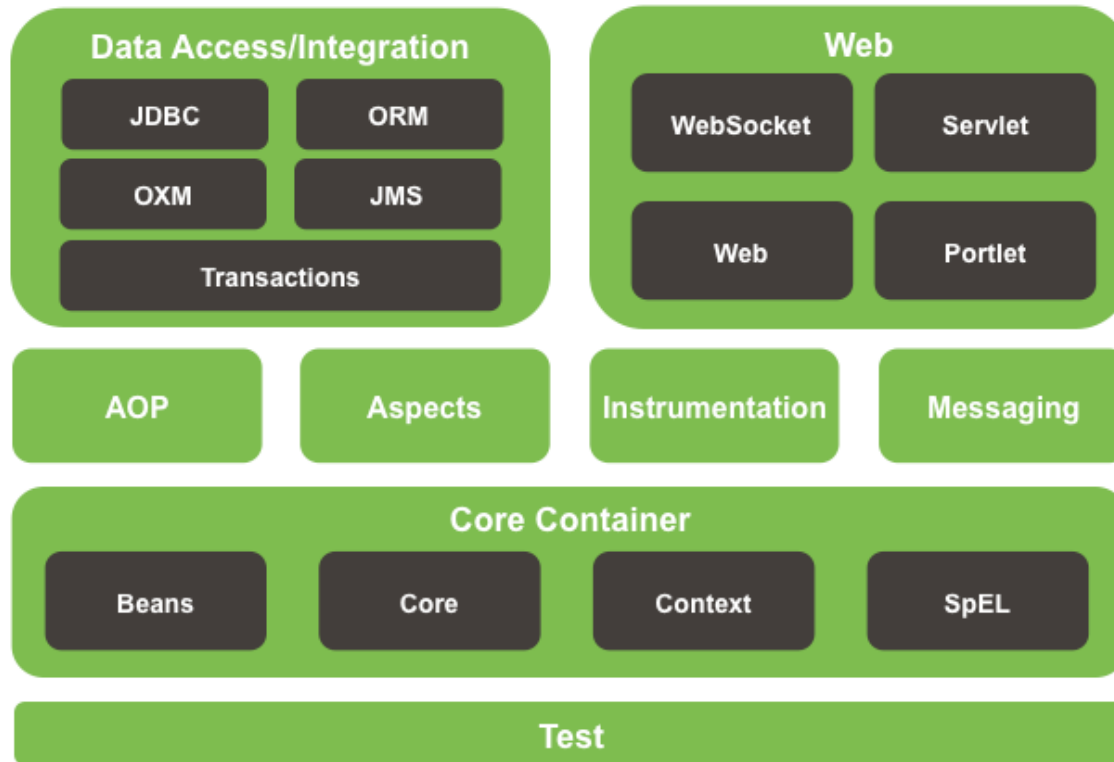


SpringBoot, Annotations et AutoConfiguration

Rappel Spring Framework



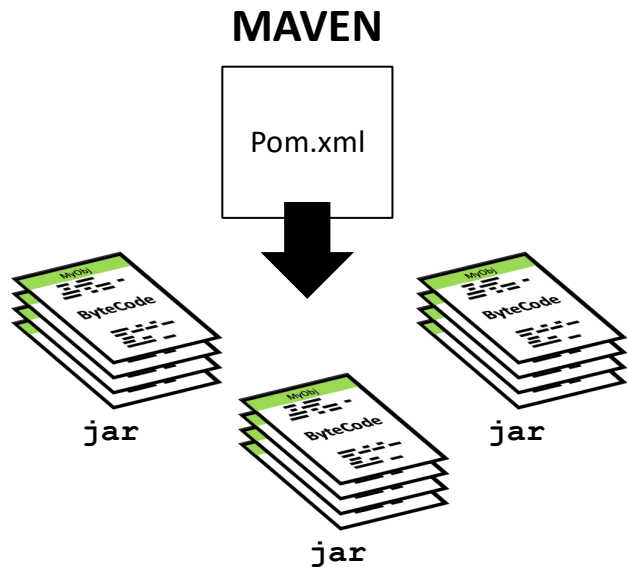
Spring Framework Runtime



<https://docs.spring.io/spring-framework/docs/4.3.20.RELEASE/spring-framework-reference/html/overview.html>

Comment ça marche ?

1 Récupération des dépendances



2 Lecture du fichier principal Springboot

```
@SpringBootApplication  
public class DemoApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(  
            DemoApplication.class, args);  
    }  
}
```

@EnableAutoConfiguration

+

@Configuration

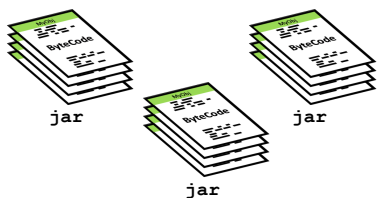
+

@ComponentScan

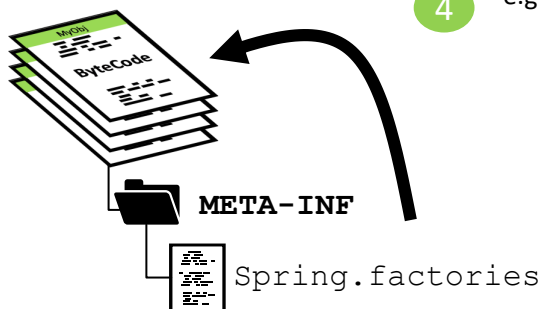
@EnableAutoConfiguration

Active l'auto-configuration du contexte d'application → génère la configuration si nécessaire (si présence de dépendances)

- 3 Sélectionne l'auto-configuration des composants présents dans les dépendances
- 4 Appel les fichiers d'auto-Configuration utilisant les annotations @Configuration, @Conditional
- 5 Créé et enregistre les Beans nécessaires à la configuration



Spring-boot-autoconfigure-x.x.x.jar



4 e.g JpaRepositoriesAutoConfiguration.class

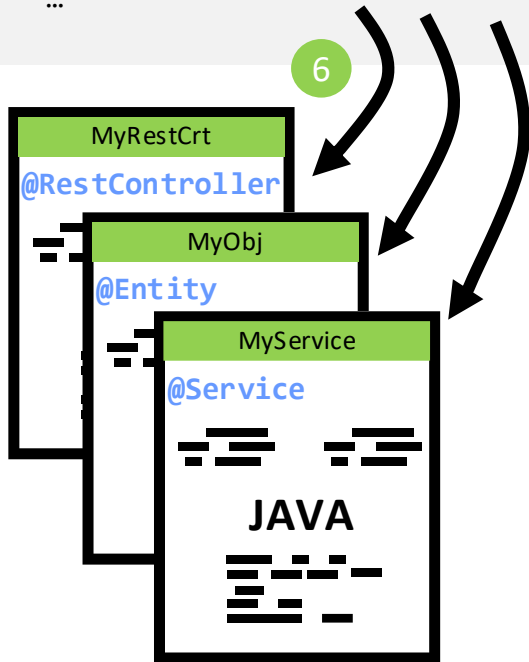
```
@Configuration(proxyBeanMethods = false)
@ConditionalOnBean(DataSource.class)
@ConditionalOnClass(JpaRepository.class)
@ConditionalOnMissingBean({ JpaRepositoryFactoryBean.class, JpaRepository...
@ConditionalOnProperty(prefix = "spring.data.jpa.repositories...
@Import(JpaRepositoriesRegistrar.class)
@AutoConfigureAfter({ HibernateJpaAutoConfiguration.class,
TaskExecutionAutoConfiguration.class })
public class JpaRepositoriesAutoConfiguration {
    @Bean
    @Conditional(BootstrapExecutorCondition.class)
    public EntityManagerFactoryBuilderCustomizer
    ...
```

@ComponentScan + @Configuration

Analyse le package donné et enregistre les Beans trouvés (analyse les @Configuration, @RestController, @Repository, @Service ...)

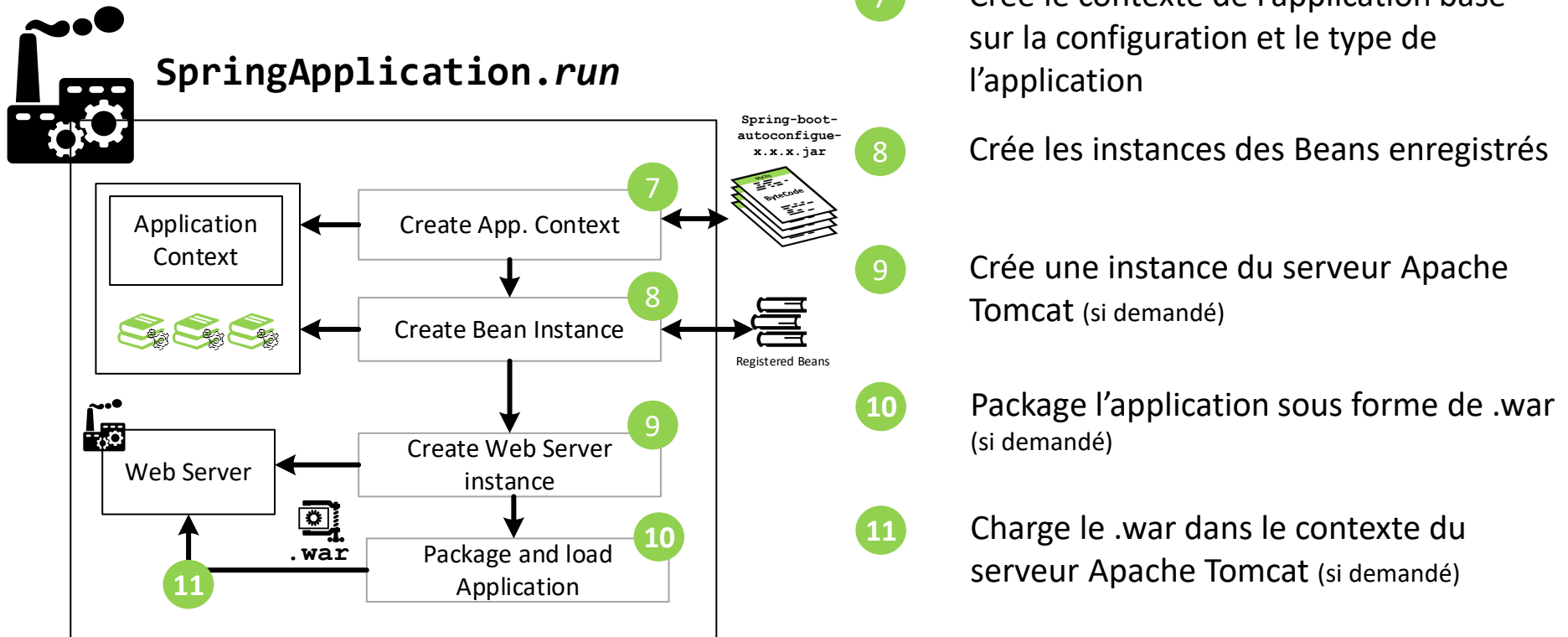
```
@SpringBootApplication  
public class DemoApplication {  
    ...  
}
```

6 Enregistre les beans du package donné



SpringApplication.run

Créer le contexte de l'application, instancie les beans enregistrés, effectue le packaging demandé.



- 7 Crée le contexte de l'application basé sur la configuration et le type de l'application
- 8 Crée les instances des Beans enregistrés
- 9 Crée une instance du serveur Apache Tomcat (si demandé)
- 10 Package l'application sous forme de .war (si demandé)
- 11 Charge le .war dans le contexte du serveur Apache Tomcat (si demandé)



Jacques Saraydaryan

Jacques.saraydaryan@cpe.fr