[Spring](http://www.springframework.org) – SpringBoot

**Antoine Bour 04/12/2016**

Contenu

[1) Spring 1](#_Toc468616376)

[a. Spring Framework 1](#_Toc468616377)

[b. Spring projects 2](#_Toc468616378)

[c. Pro and cons 2](#_Toc468616379)

[d. Spring Core 2](#_Toc468616380)

[i. Definitions 2](#_Toc468616381)

[ii. Initialization 3](#_Toc468616382)

[iii. Property files 3](#_Toc468616383)

[iv. Namespaces 3](#_Toc468616384)

[v. Spring container 3](#_Toc468616385)

[vi. Conf file 4](#_Toc468616386)

[vii. Inversion of Control 4](#_Toc468616387)

[viii. Annotations 7](#_Toc468616388)

[2) Spring Boot 11](#_Toc468616389)

[a. Why? 11](#_Toc468616390)

[b. Exemple: 11](#_Toc468616391)

[c. Starters: 13](#_Toc468616392)

[d. Executable jar. 14](#_Toc468616393)

[e. Spring Boot feature 16](#_Toc468616394)

[f. Developing Web applications: 17](#_Toc468616395)

[g. Security: 17](#_Toc468616396)

[h. Databases (Topic already seen for the assignment): 18](#_Toc468616397)

[i. Caching @CacheResult; component of Spring Framework 18](#_Toc468616398)

[j. JMS / Refer to Spring Framework 18](#_Toc468616399)

[k. Other: 18](#_Toc468616400)

# Spring

* Development application framework, especially for enterprises
* Inversion of control container
  + Lightweight container
  + Bean lifecycle management
  + Injection of dependences
  + Aspect oriented programmation
  + Lot of API linked
  + Declarative transaction management
  + DAO for persistence is easier
  + Integration with different layer is easy
    - Spring is usually used in order to create and inject the required object from the precedent layer

## Spring Framework

|  |  |  |
| --- | --- | --- |
|  | **Module** | **Description** |
| Spring Framework 3 | Spring Core container | Container building |
| AOP and instrumentation | AOP building |
| Data access/Integration | Access to the datas |
| Web | For web development |
| Test | For automated tests with Spring |
| Spring Core Container | Spring Core and Spring Beans | Base functions: Container and tools |
| Spring Context | Configuration, mail, JNDI, Internationalisation |
| Spring Expression Langage | Langage of expression in order to use the objects generated by the container |
| Spring AOP and Instrumentation | Spring AOP | AOP support (more details are provided in this doc) |
| AspectJ | AspectJ integration (more details are provided in this doc) |
| Instrumentation | Class instrumentation and classloader used by Applications Servers |
| Data Access / Integration | Spring JDBC | Abstraction of JDBC Use with exception hierarchy |
| Spring ORM | Support for JPA, JDO, Hibernate… |
| Spring Transaction | Transaction handling |
| OXM | XML Mapping with JaxB Support … |
| JMS | JMS support with Spring |
| Web | Spring Web | Web development (container, context, multipart support, HTTP parameters) |
|  | Spring Web-Servlet | Development Framework for MVC concept. Allows to choose the framework used for the view (JSP, …) |
|  | Spring Web-Struts | Struts Support |
|  | Web-Portlet | Portlet support |
| Test | Spring Test | Automated tests (Junit, TestNG..) |

## [Spring projects](https://spring.io/projects)

Plugins that facilitates some applications functionalities:

Example:

|  |  |
| --- | --- |
| Spring Web Services | Develop SOAP web services |
| Spring Security | Acegi Security, for authentication and permission in a web app |
| Spring batch | For batch applications |
| Spring LDAP | For LDAP use |
| Spring IDE | For integration in Eclipse |
| Spring Data | For NoSQL use |
| Spring Android | Develop for Android application (REST/Auth) |
| Spring boot | More details further in the doc |
| … |  |

## Pro and cons

* Open Source project
* High integration of open source framework (Struts, Hibernate) and Java projects
* All functions are adapted to a JEE server use.
* Complete documentation
* Not always easy to use.
* Generated code can be important due to a lot of dependences required by Spring

## Spring Core

### Definitions

* Beans

Object that form the backbone of the application and that is managed by the Spring IOC container. This objects is instatiated, assembled and managed by a Spring IOC Container.

There is a scope for all beans:

* + Singleton: The container can only have 1 instance for the bean. On each call the same instance is returned
  + Prototype: On each call a new instance is provided.

The bean tag defines a bean. The following attributes are possible:

|  |  |
| --- | --- |
| Attribut | Role |
| Abstract | Define if the bean is abstract. True if the container should not create an instance |
| Autowire | Define how the bean will be injected |
| Class | The name of the bean class |
| Id | Bean identifier |
| Init-method | Called initizialisation method which is called once the bean instanced and the dependences injected |
| Scope | Singleton, prototype, request, session |
| … |  |

Callbacks: PostConstruct, PreDestroy, afterPropertiesSet: Method are called based on Bean lifecycle.

### Initialization

By Default: Singletons are initialized on container initialization.

Initialization can be saked only one first use by using the attribute lazy-init of the bean tag.

### Property files

Property can be exported in a property files. In this case, only use placeholder ${…} for resolution

### Namespaces

Their goal is to simplify the definition of the context (aop,jee,lang,util..)

**Jee** simplify the configuration of a JEE environment by getting a JNDI object or an EJB reference per example.

**lang** in order to configure scripting langages object in the context (Groovy,Jruby,BeanShell…)

**util,** facilitates the configuration of list, map, set in a bean.

…

You own namespace can be defined

### Spring container

Org.springframework.context.ApplicationContext represent the IOC container. It is responsible for instantiating, configuring, assembling the beans. The container get its instruction (what object to instantiate, configure, assemble) by reading configuration metadata (XML, annotation or javacode)

In order to load the container:

BeanFactory factory = new XmlBeanFactory(new FileSystemResource("c:/beans.xml"));

XMLBeanFactory factory = new XMLBeanFactory(new ClassPathResource ("appContext.xml"));

### Conf file

* Xml document with <beans> root tag.
* Only the transversal objects have to been defined or the technical components that requires a specific configuration.
* Can be splitted in multiple files (import them with relative or explicit path)

### Inversion of Control

**With IOC:**

Objects define their dependencies, the objects they work with. This is done only through constructor arguments, arguments to a factory method or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean.

*org.springframework.beans* or *org.springframework.context* are the basis of IOC.

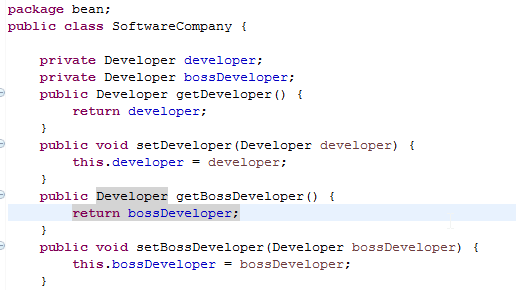
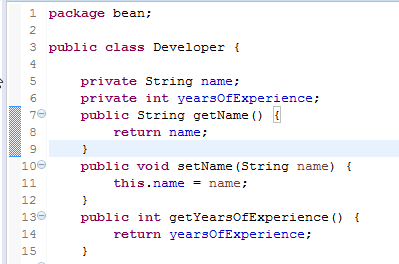
BeanFactory interface provides an advances Configuration mechanism capable of managing any type of object (ApplicationContext is a sub-interface of it). It adds integration with AOP features, message resource handling, Specific contexts (WebApplicationContext)….

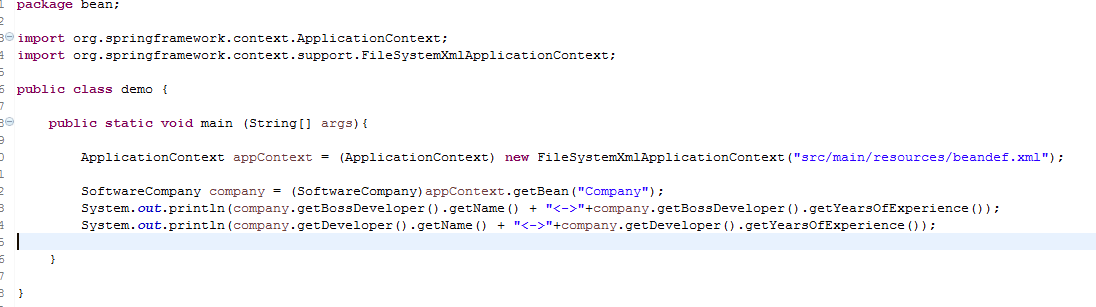
**Without IOC:**

The bean control the instantiation or location of its dependencies by using direct construction of classes or mechanism like the Service Locator pattern

**Dependences Injections**

* Constructor&

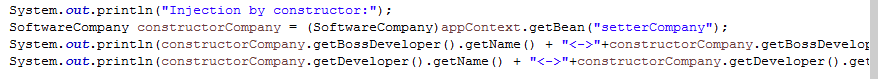


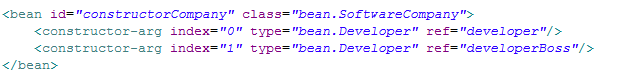




There are 2 possibilities in Spring:

* Setter injection : Refer to the below exemple
* Constructor injection:





**Which one to use?**

“By constructor”: the instance is fully initialized after it’s instanciation. Not to do if they are too much dependencies, or if some dependences are optional. Be careful to of circular dependences.

« By setter » allows modifying the instance of the dependences.

Both choices can be mixed. Their uses depend of the context.

* [Autowiring](https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm)

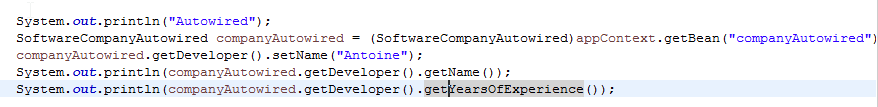
The container determines automatically the managed object to inject as a dependences in an another object. This simplify the configuration file: It become obsolete to define parameters value for setter or constructor when you inject a dependence.

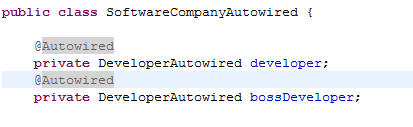
Autowiring is working following differents strategies:

|  |  |
| --- | --- |
| No | No autowiring. Dependences have to be explictely described in the configuration |
| byname | Inject automatically a property based on its name. The container search in the generated objects a unique object with the same identifiant. If found, the container inject the object |
| byType | Inject automatically a property based on it’s type. The container search in the generated objects a unique type from same type than the property. If found, the container inject the object. |
| Constructor | Same as byType but using the constructor parameters |
| Autodetect | Use constructor or byType mode after analysis of the bean class. |

* The default-autowire attribute of bean tag let know the mode used.
* An explicit definition is always prior to autowiring
* Can only be used for objects. Impossible to use for Primitives types, String, table…
* Simplify the bean configuration and maintain it up to date in case of new dependence in a bean if the bean uses autowiring.
* IDE can be lost because the dependances are not explicitely defined
* Autowire-candidate to false say that an object is not candidate to autowiring.
* It can work with dedicated annotations

Exemple :



It can be done on property level, constructor level, setter methods.

* [Spel](http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/expressions.html)
  + Spring Expression Language
  + Allows object manipulation on runtime
  + Facilitates configuration
  + Supports regex, operators, collections manipulations…
  + Access to class properties, methods invocations, constructor invocations..
  + Can be used in Spring configuration file, annotation @value, JSP,…
  + Cool in order to access /write system properties (DB config, JVM param..)

### Annotations

Usefull for configuration, the Spring config file is easier to read. The configuration is simplified and decentralized. The container has to know that we user annotations:

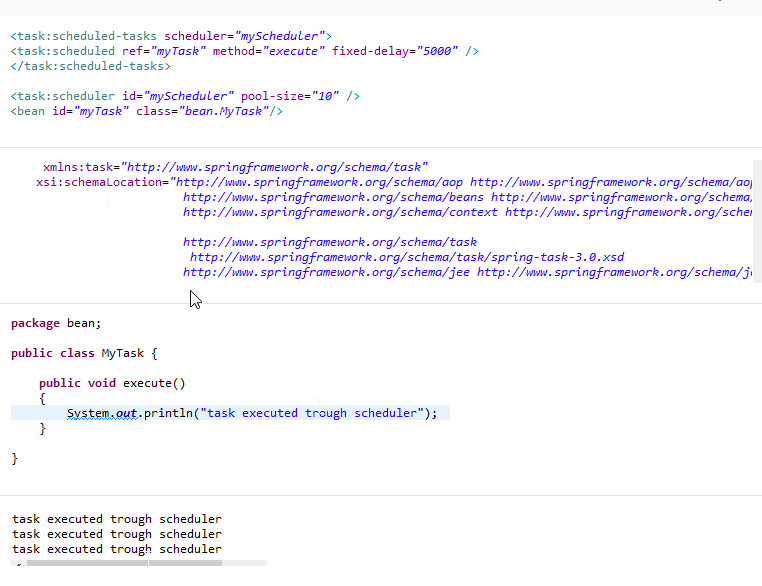


|  |  |
| --- | --- |
| Annotation name | Description |
| @scope | Define the bean lifecycle (singleton, prototype, session, request) |
| Injection of Dependances: @autowired | Ask for an automatique type injection |
| Injection of Dependances: @configurable | Determine the bean to inject if multiple container instances are corresponding to the wished type |
| Injection of Dependances: @qualifier | For a class with dependences managed by the container. In this case, the dependences are injected when the constructor is called. Requires AOP with AspectJ. Used with autowired. Qualifier define the exact bean to inject (if more bean with same type are defined) |
| Injection of Dependances: @resource | Automatical injection by name |
| @required | Validate the injection of the dependances. If the property is null it generate an Exception. Usefull in order to avoid NullPointerExceptions |
| @Autowired | Refer this doc |
| @Resource | Ask bean injection based on its name. Used on a field or a method. Autowired makes resolution by Type. Resource makes resolution by name. |
| @Configurable | Used by AOP to intercept the invocation of a constructor and inject class dependence’s |
| @Bean | To use for a method that create a bean instance  Attribute:  Init-method for the method name called on the initizialisation  Destroy-method for the method called when the bean is pushed out of the container  …. |
| @DependsOn | Container creates first indicated dependences |
| @Lazy | Only initialized when a bean require it |
| @Inject | Same as @Autowired. But with inject the injection has to be on an instance, there is no optional parameter. |

[Should I use @inject, @Resource or @Autowired?](http://blogs.sourceallies.com/2011/08/spring-injection-with-resource-and-autowired/)

* [LifeCycle](http://javabeginnerstutorial.com/spring-framework-tutorial/java-spring-bean-lifecycle/)
* Scheduling:

It is possible to schedule tasks through Spring: Refer to my git project for an exemple:

**

*It’s possible to schedule a task by using the @Scheduled(fixedDelay=5000) annotation (over the metho execute).*

* 1. Aspect Oriented Programmation
     1. Concept

AOP (AspectJ or Spring AOP) proposed transversal or specific functionalities (transaction handling), in a declarative way (annotation or XML). This functionalities are named **advice** and are executed when events, named **joinpoint** (ex: call of a constructor, a method…) are met.

The place where the **advices** are called when the **joinpoint** are met is defined through **pointcuts.**

AOP is interesting to put in place transversal technical functions like transactions. Spring is managing transactions with AOP. AOP is used in the Spring Mechanism itself, for @Configurable by example.

AOP can be used for the developer own’s aspects. (Refer to my git example).

There are 2 ways to put AOP in place:

* AspectJ: Runtime or compilation
* [Spring AOP](http://howtodoinjava.com/spring/spring-aop/spring-aop-aspectj-example-tutorial-using-annotation-config/): Spring solution based on dynamically created proxies.

Less powerful than AspectJ, only on public methods, only possible to define it in the application context…

Let’s focus on AspectJ: Please take a look at the git project for an easy example.

* 1. Transaction Handling with Spring

Based on abstraction that allows a generic way to put them in place without taking care of the technology used (JTA / JDBC / JPA / Hibernate)

So, Spring proposes different transaction manager: For Hibernate, JTA, JDBC…. The use of transactions is the same, only the declaration of the manager is describing the way to handle them

A transaction handled by Spring has the following characteristics:

|  |  |
| --- | --- |
| Isolation | Describe the isolation level of a transaction for other transactions |
| Propagation | How the processes are integrated in a transactional context |
| TimeOut | Maximum time allowed to execute the transaction. After this time: Rollback |
| Read Only | Describe if the data can be read only or updated.. Useful for optimization |

A transaction can be logic or physic:

|  |  |
| --- | --- |
| PROPAGATION\_REQUIRED | Creates a logical transaction for each method in the transactional context |
| PROPAGATION\_REQUIRES\_NEW | Create a new independent transaction. No impact on the parent transaction |
| PROPAGATION\_NESTED | Physical transaction with savepoints |

Transactions are defined in the context through a specific TransactionManager. It’s necessary to define the configurations of aspects for transactional methods. They are defined through the jointpoint thanks to templates associated to advice.

By the way, AOP can be used for transaction and they can be used with annotations:

Please refer to my git project: An issue in the second insert (pet) is causing a whole transaction rollback.

# [Spring Boot](http://docs.spring.io/spring-boot/docs/2.0.0.BUILD-SNAPSHOT/reference/html)

## Why?

* + Spring is too much complicated
  + No config files here, starting a project is going very quickly
  + Facilitates the access to the spring env (security, MVC)
  + It configures Spring for me (in most of the case)
  + Spring boot handles your dependences (stand-alone application)
  + Spring boot handle your deployment
  + Spring contains a WebContainer (embed Jetty, Tomcat)
  + Spring boot manages your web app and provides metrics.
  + No hot deployment
  + Principle “Convention over configuration”
  + Only need to concentrate on your dev
  + Provide metrics.

## [Exemple:](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started)

Git: <https://github.com/Assignment2016/fiveminSpringBoot-.git>

Explanation:

* Pom.xml
  + Include spring-boot dependencies: spring-boot-starter-web
  + Point to the spring boot repository (not necessary for a RELEASE version use of Springboot).
* There is only 1 dependence to declare
* Samplecontroller:
  + Controller
  + The list of available bean is scanned thanks to @ComponentScan.
  + Spring boot require a main method to run. It’s our entry point.
  + Application can be started with mvn spring-boot: run
  + @Component indicates that the class is a component. They are candidate as auto-detection for the ComponentScan.
  + @Controller: Spezialisation of component. It is scanned with @ComponentScan and indicates that the page is a (web) Controller. @RestController is a controller

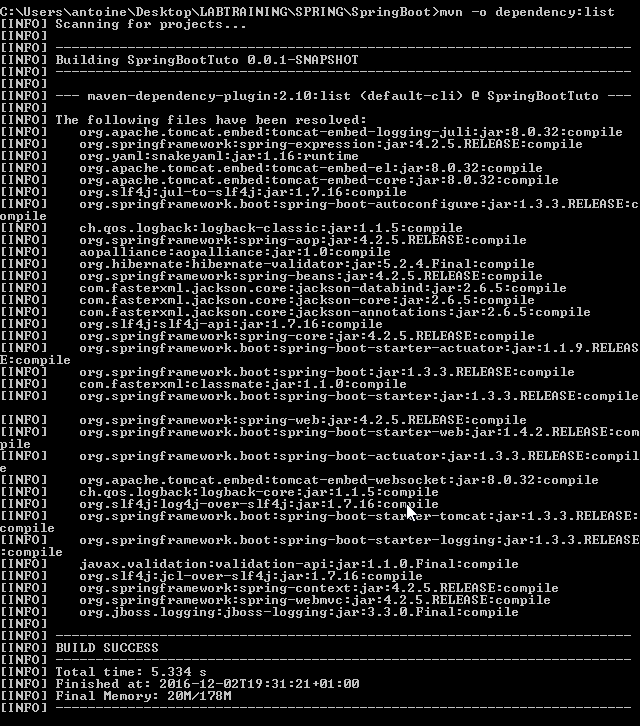
Providing routing information. This is only a Stereotype annotation providing hits for people reading the code and for Spring, it says that the class plays a specific role.

In our exemple, just change @Controller to @RestController.

* + @EnableAutoConfiguration : For the Spring application Context, attempting to guess and configure beans that the application is needed. More setup is possible. It’s possible to exclude bean. It has to be place in the root class of the project
  + @Autowired: Link the bean to our class instance.
  + @responseBody: Indicates that the response returned should be bound to the web response body (text/json..)
  + @[RequestMapping](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html): Mapping the web request (and associated params) onto specific handler (method)
* IDisplayer: Just the interface for the service Implementation
* DisplayerImpl: Contains a Function that returns a String.
* The dependency spring-boot-starter-actuator allows to join some URL that manage the application:
  + <http://localhost:8080/health>
  + <http://localhost:8080/metrics>
  + <http://localhost:8080/beans>
  + <http://localhost:8080/dump>
  + …
* We have included 2 dependencies here, but what about the nested dependences:

Mvn dependency:resolve

Mvn –o dependency:list



* This light example has been built and deployed in less than 5 minutes!
* Create an executable jar : <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-first-application-executable-jar>

## [Starters](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot-starter):

Like a mvn archetyp, spring provides starters that make easy to add jar s to your classpath:

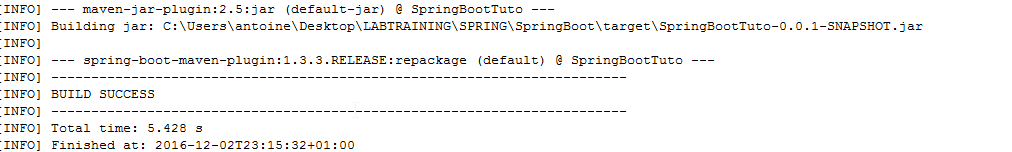
* + Spring-boot-starter-parent: provides useful Maven defautls and dependency-management section so that you can omit version for nested dependencies.
    - Springboot dependencies is like a **bom** file. It manages my maven dependencies
    - Bypassing the BOM is possible by using <scope>Import</scope> for each dependency
  + Other starters handle the dependences for the wished type of developed application: - Web application: spring-boot-starter-web
    - * + In order to test, just check mvn whole list of dependencies after having added this as mvn dependencies

- The [most popular starters](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters) are:

Web Starter, Test Starter, JPA starter, Mail Starter and spring-boot-starter-actuator for adding production ready features. We will see this last starter after.

### Executable jar.

* We can run it in production
* The jar contains all dependencies. (java –jar \*\*\*.jar /// -Xdebug for debug)
* Only add spring-boot-maven-plugin as a build-plugin in pom.xml
* Run “mvn package”



* Lot of sample [here](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples)
* Using Spring Boot:
  1. <http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot>
  2. A build system that supports dependency managementis required
  3. The build system has to be able to consume artifacts from maven.

Conclusion: Use maven or gradle.

Spring placeholder: ${..}

Maven placeholder replaced to @..@ (override is possible with maven property resource.delimiter)

* 1. Technical starters are existing:

|  |  |
| --- | --- |
| spring-boot-starter-undertow | Alternative to spring-boot-starter-tomcat |
| spring-boot-starter-jetty | Alternative to spring-boot-starter-tomcatg |
| spring-boot-starter-logging | Default logging starter |
| spring-boot-starter-tomcat | Default container starter used in spring-boot-starter-web |
| spring-boot-starter-log4j2 | Alternative to default loggin starter |

* 1. Structure of the code. Best practices :
* Do not use the default package
* Respect Java recommended package naming (com.example.project)
* @enableAutoConfiguration on main class, same for [@Configuration](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot-configuration-classes) (if you do not use enableAutoConfiguration
* Recommanded layout:



* @ImportResource can be used to import XML configuration
* Auto Configuration is not invasive:

Defining your own specific bean for DBA will remove all the default configuration for DB provided by the enableAutoConfiguration. DataSourceAutoConfiguration or other specific configuration can be excluded with parameter “exclude “ in @EnableAutoConfiguration

* @SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan. This is a encouraged practice.
  1. For Dependency Injection, all Spring framework techniques are allowed. (Refer to Spring doc aboved)
  2. [Dev tools](https://www.youtube.com/watch?v=A70NMxV13TI) exists for a better developing experience:
     + Java code update detected by Spring Boot. The application restart dynamically
     + Change in html/css do part of code do not require a browser refresh. This is automatically done by Spring Boot
     + Remote debugger
     + Pushing change on remote started application

### Spring Boot feature

* The startup process can be updated to contain your own text, banner.
* It’s possible to redefine the SpringApplication process by creation a local instance of SpringApplication (refer to the git project attached)
* SpringApplication can be update to chain your method calls and call parent methods or child methods in order to create your own hierarchy
* Creation of listener is possible (on failed starting, success starting, ready…)
* ApplicationRunner or CommandLineRunner allows to run specific code once the SpringApplication is starter.
  + RandomValuePropertySource is useful for injecting random values (refer to associated project)
  + Beans can implement ExitCodeGenerator interface in order to return a specific exit code once the application ends.
  + Remote administration is possible with spring.application.admin.enabled property

In order to manage the same application in differents environment it’s possible to externalize the configuration:

* + Property are:
    - Global properties file on home directory
    - Test annotation @TestPropertySOurce
    - Init parameters from ServletConfig or ServletContext
    - JNDI attributes
    - Java System properties
    - ….
  + Properties are in application.properties
  + Or provided as parameter on jar starting
  + SPRING\_APPLICATION\_JSON can be supplied on Command Line.
    - This JSON contains a list of Key: values
  + Profile properties are allowed: application-{profile}.properties
  + User of placeholder is possible too.
  + Using YAML (superset of JSON) instead of properties is possible
* Logging
  + Datetime precision: Millisecond
  + Log level are DEBUG / INFO / TRACE / WARN / ERROR
  + Console output provided by default ERROR / WARN and INFO
  + –DEBUG on start to add debug level
  + Colored output is possible (if terminal supports ANSI). FATAL and ERROR = red, WARN = YELLOW, else it’s GREEN
  + Use logging.file or logging.path properties in application.properties to log to file.
  + Log level can be adapted for wished framework:
    - Logging.level.org.springframework.web
    - Logging.level.org.hibernate
  + Logging can be adapted to wished environment based on <springPRofile> tag in a logback xml file.

### Developing Web applications:

* As seen in our git project, an embedded HTTP server is embedded in SpringBoot (spring-boot-starter-web)
  + Spring boot provides support
    - Servlet registering  
      Servlet Context initialization
* @JsonComponent helps to (de)serialize JSON datas
* Static content is served by Spring Boot (/static folder). This can be adapted by updating spring.resources.staticLocations
* Src/main/webapp works only with war file. It is ignored for a jar.
* Dynamic HTML can be used (Server side rendering).. but JSPs are not rated.
* ErrorMapping is provided
  + Build of own pages depending on HTTP error (404…)
* Cross Origin Resource Sharing support is provided (@CrossOrigin)

### Security:

* Is Spring Security on the classpath then all HTTP endpoints will require authentication
  + @EnableGlobalMethodSecurity
    - Indicate role to wished methods:
      * IS\_AUTHENTICATED\_ANONYMOUSLY
      * ROLE\_TELLER
      * ….
* The default AuthenticationManager has a single user that is displayed on application starts-up (SecurityProperties.user)
* All security check can be switched-off or modified
* [OAuth2 support](http://projects.spring.io/spring-security-oauth/docs/oauth2.html): Exemple with SSO: <https://spring.io/guides/tutorials/spring-boot-oauth2/>
* The Spring Boot documentation is not helpful in order to understand a login mechanism, please refer to <http://docs.spring.io/spring-security/site/docs/current/reference/html/jc.html>

### Databases (Topic already seen for the assignment):

* Spring boot can manage embedded databases (h2,HSQL, Derby)
* Lot of differents Database Frameworks are supported

### [Caching](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-caching) @CacheResult; component of Spring Framework

* + Many providers are supported:
    - Generic
    - JCache
    - Hazelcast
    - Redis
    - Caffeine

### [JMS](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-jms) / Refer to Spring Framework

### Other:

* + - * REST Services (See Assignment)
      * A Validation feature is provided for method parameters
      * Emailing is also possible through JavaMailSender
      * Testing
        + Support of Junit, Mockito,….
      * Spring boot CLI
        + Allows to run groovy scripts
        + Allow to run Spring boot applications

Spring run \*\*\*.groovy

Package the application

Initialize a new project

* + - * Packaging the app
      * [Initialize a new project](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#cli-init)