|  |  |
| --- | --- |
|  | **Spring Core - University program** |
| Author: GTN | Document type: Self-training roadmap |
| Review: QVR |  |
| Date: 01.06.2023 |  |
| Version: 1.0 |  |

Self-Training for Spring Core

**Table of contents**

1. Introduction 1

2. Prerequisites 3

3. References 4

4. Glossary 4

5. Training plan 6

5.1. Basic 6

5.2. Advanced (optional) 11

# Introduction

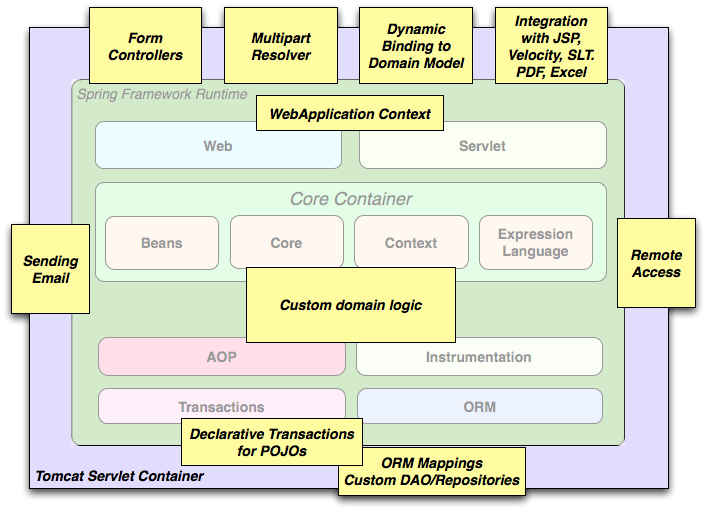
Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform. A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.



Among these module, Web MVC is a framework written by Spring as a reaction to what they perceived as poor design of the popular Jakarta Struts web framework, as well as deficiencies in other available frameworks [SPRING-MVC-WIKI].

Spring Framework and Spring MVC is currently used widely in many Java-based projects in ELCA (typically in a full-fledged scenario as below). Therefore, having good knowledge about them is a must to do your job effectively.



# Prerequisites

Having basic knowledge about:

* Java
* Request / Response paradigm
* MVC pattern

Having the following software installed:

* Java SDK 8+
* Maven (3.2.5+)
* IntelliJ 2020+

# References

|  |  |
| --- | --- |
| **Ref** | **Source** |
| [SPRING-IN-ACTION] | Craig Walls. Spring In Action, 4rd Edition, Manning Publications 2014 |
| [JSR-000315] | <http://download.oracle.com/otndocs/jcp/servlet-3.0-fr-oth-JSpec/> |
| [SPRING-MVC-WIKI] | <http://en.wikipedia.org/wiki/Spring_MVC#Model-view-controller_framework> |
| [SPRING-WIKI] | <http://en.wikipedia.org/wiki/Spring_framework> |
| [WEBAPP-FW-WIKI] | <http://en.wikipedia.org/wiki/Web_application_framework> |
| [CASESTUDY] | Appendix A |
| [REST-WIKI] | <https://en.wikipedia.org/wiki/Representational_state_transfer> |

# Glossary

|  |  |
| --- | --- |
| **Term** | **Description** |
| Servlet | [JSR-000315] A servlet is a JavaTM technology-based Web component, managed by a container, that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container. |
| Servlet container | [JSR-000315] The servlet container is a part of aWeb server or application server that provides the network services over which requests and responses are sent, decodes MIME-based requests, and formats MIME-based responses. A servlet container also contains and manages servlets through their lifecycle. |
| REST | In computing, Representational State Transfer (REST) is the software architectural style of the World Wide Web. REST gives a coordinated set of constraints to the design of components in a distributed hypermedia system that can lead to a higher performing and more maintainable architecture. |

# Training plan

## Basic

These steps are mandatory and should be completed by all trainees.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Type[[1]](#footnote-1)** | **Task** | **Description** | **Estimated Time (h)** |
|  | R | Acquire general concept about Spring Framework | Get an “Overview of Spring Framework” here:  <https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/htmlsingle/#overview>  *Notes: The reading can stop at section 2.3.1 (Dependency Management and Naming Conventions) which is not necessary for this training.* | 1 |
|  | R | Acquire basic understanding of Spring framework | Read the following chapters of the book [SPRING-IN-ACTION]:   * Chapter 1: Springing into action   + Section 1.1.1, 1.1.2 (DI, Wiring, Choose Implementations)   + Section 1.2.2 (Lifecycle of beans) * Chapter 2: Wiring beans   + Section 2.1, 2.2, 2.3 * Chapter 3: Advanced wiring   + Section 3.1, 3.2, 3.3   Read the following link to understand how Java based configuration can be used to configure a Spring application without XML: <http://www.tutorialspoint.com/spring/spring_java_based_configuration.htm>.  You are expected to know at least:   * Spring’s core concepts:   + Dependency injection:   Manage and inject dependencies into objects, promoting loose coupling and modular design.  Decoupling Dependencies, Inversion of Control  Types: Contrustor Injection, Setter Injection, and Method Injection.  Benefits of DI in Spring: DI promotes loose coupling, making code more modular, reusable, and testable.  Spring provides the @Autowired annotation to simplify dependency injection. It automatically wires dependencies by type or name.  Other annotations like @Qualifier can be used for more fine-grained control when multiple dependencies of the same type exist.   * + Spring bean life cycle   **Instantiation**: The Spring container creates an instance of a bean (constructor).  **Dependency Injection**: the container injects any dependencies into the bean  **Initialization** This phase involves executing any initialization callbacks, such as the @PostConstruct annotated method or implementing the InitializingBean interface.  **Ready for Use:** Once the initialization phase is complete, the bean is ready for use.  **Runtime Operations:** performs its designated tasks and interacts with other components in the system.  **Destruction**: This typically occurs during application shutdown or when the bean is no longer needed.  **Destruction Callbacks:** Before the bean is destroyed, any destruction callbacks are invoked, such as the @PreDestroy annotated method or implementing the DisposableBean interface.  **Garbage Collection**: Finally, the bean instance becomes eligible for garbage collection by the Java Virtual Machine (JVM).   * How to configure and wire Spring beans?4   **Config :**  *XML Configuration* in applicationContext.xml (<bean>, <property>)  *Annotation-Based Configuration*: @Component, @Autowired, @Qualifier  *Java Configuration*: These classes are annotated with @Configuration, and bean definitions are created using @Bean annotations. Java configuration provides a more type-safe and concise approach compared to XML.  **Wiring**  *Component Scanning*: automatically detects classes annotated with @Component ,@ Service, @Repository, etc.  Spring can automatically create and wire beans without explicitly defining them.  *Dependency Injection:* directly injecting dependencies into fields using annotations.  *Qualifiers*: When multiple beans of the same type exist, you can use the @Qualifier  *Auto-Wiring:* Spring provides auto-wiring capabilities, where it automatically detects dependencies and wires them based on the defined rules. Auto-wiring can be enabled using annotations like @Autowired, @Resource, or through XML configuration.   * How autowiring and autodiscovery work?   **Autowiring:**  Autowiring is a mechanism in Spring that automatically resolves and injects dependencies into beans.  When autowiring is enabled, Spring scans the application context for beans that match the required dependency type. If a single matching bean is found, it is injected into the dependent bean. If multiple matching beans are found, Spring tries to resolve the ambiguity based on qualifiers, bean names, or other rules.  **Autodiscovery:**  Autodiscovery, also known as component scanning, is a feature in Spring that automatically detects and registers beans based on certain conventions.  Spring scans the application's classpath and identifies classes that are annotated with specific annotations like @Component, @Service, @Repository, etc.  During autodiscovery, Spring identifies these annotated classes and creates bean instances for them.   * Autodiscovery is responsible for finding and registering components, beans, and configuration classes in your application, while autowiring is responsible for automatically connecting these components together by resolving and injecting their dependencies. | 4 |
|  | R | Understand Spring MVC | Read the the following sections from the book [SPRING-IN-ACTION]:   * Chapter 5 (full)   Please also read the following sections from Spring’s reference to understand some commonly used annotations that are not covered in the book:   * @ModelAttribute * @InitBinder | 4 |
|  | R | Understand Spring MVC REST | In recent years, REST has emerged as a popular information-centric alternative to traditional SOAP-based web services. Read the following sections from the book [SPRING-IN-ACTION] to understand the REST support from Spring MVC:   * Chapter 16 (full)   + @RequestMapping / @RequestParam   + @RequestBody / @ResponseBody | 1.5 |
|  | E | Setup development environment | Check out the project pim-tool-back from Git (link should be given by your coach).  There are 2 ways to compile the project code:   1. **Command line**   Configure file etc/env/setenv.bat   * Set **JAVA\_HOME** to the directory where Java is installed * Set **M2\_HOME** to the directory where Maven is installed   Copy etc/maven/settings.xml to your local M2\_HOME\conf.  Bootstrap the project with Maven commands:   * From the directory pim-tool-back, open command line, type ..\etc\env\setenv.bat. * Verify versions of Java and Maven by executing the following commands:   + java –version   + mvn -version * Type mvn clean install to clean and build the whole project. In this step, missing dependencies will be downloaded to the local repository and you should see “BUILD SUCCESS” in the command line at the end.  1. **IntelliJ**   Configure *Project Structure* and *Settings* to use your installed Maven and Java versions.      Toggle “Skip tests” mode on.  Run “Clean” and “Install” steps from lifecycle. In this step, missing dependencies will be downloaded to the local repository and you should see “BUILD SUCCESS” in the command line at the end. | 1 |
|  | E | Postman | Postman is a API client used to test backend application.  <https://www.postman.com/>  Import the following collection and use the requests to test your Spring application in the next exercises | 0.5 |
|  | E | Working with the Spring configuration in the project | In this step you are expected to understand clearly how Spring was applied.   * Open ApplicationLauncher and start it. * TaskRepository implementation has wrong name. Therefore Spring cannot find and wire it correctly. *Can you find that class and rename it?* * From Postman, send a request to <http://localhost:8080/>main and you will see NullPointerException (actually 3 NullPointerException(s) respectively). *Can you find the causes and fix them?* | 2 |
|  | E | Complete the query-by-name on “search” feature | Make the necessary changes in pim-tool-back to have an GET endpoint that receive a “keyword”. The response must return a list of projects with name containing the keyword.  *Notes:*   * *The implementation of ProjectServiceImpl must be used.* * *If you’re asking yourself how the application works without any “explicit” Spring MVC configuration (i.e. DispatcherServlet, HandlerMapping, ViewResolver, …), don’t worry. We’re using Spring Boot for convenience and the framework is capable of configuring a Spring MVC Web application automatically for you. More information can be found here:* [*http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html*](http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html) | 2 |
|  |  | Implement new endpoints | In pim-tool-back, add new features:   * GET endpoint to find a project by its ID and return the following information: ID, name, customer, finishing date. * POST endpoint to update a project, the following fields can be updated: name, customer, finishing date. Date format must be “dd/MM/yyyy”   *Notes*:   * *The dummy implementation of ProjectService must be used.* * *Postman: create new requests to prove your work.* | 4 |
|  | C | Check | Coach checks the understanding of trainee about the way the following annotations work:  @Configuration  is used in Spring to indicate that a class is a configuration class.  A configuration class contains bean definitions and other configuration settings for the Spring application context.  @Bean  @Bean đánh dấu method cho biết rằng phương thức đó sẽ trả về một đối tượng được quản lý bởi Spring IoC container  It is used to explicitly declare individual bean definitions within a Java configuration class. The method annotated with @Bean is responsible for creating and configuring a bean instance.  @Component  to indicate that a class is a Spring component  @Component là một Annotation (chú thích) đánh dấu trên các Class để giúp Spring biết nó là một Bean.  Trong quá trình dò tìm này, khi gặp một class được đánh dấu @Component thì nó sẽ tạo ra một instance và đưa vào ApplicationContext để quản lý.  @Component gắn cho các Bean khác.  Kiến trúc Controller-Service - Repository chia project thành 3 lớp:  Consumer Layer hay Controller: là tầng giao tiếp với bên ngoài và handler các request từ bên ngoài tới hệ thống.  Service Layer: Thực hiện các nghiệp vụ và xử lý logic  Repository Layer: Chịu trách nhiệm giao tiếp với các DB, thiết bị lưu trữ, xử lý query và trả về các kiểu dữ liệu mà tầng Service yêu cầu.  Về bản chất @Service và @Repository cũng chính là @Component. Nhưng đặt tên khác nhau để giúp chúng ta phân biệt các tầng với nhau.  @Controller  @Controller là nơi tiếp nhận các thông tin request từ phía người dùng. Nó có nhiệm vụ đón nhận các yêu cầu (kèm theo thông tin request) và chuyển các yêu cầu này xuống cho tầng @Serivce xử lý logic.  @Controller is typically used for traditional MVC-based web applications, where the controller handles requests and returns the view or ModelAndView  @RestController  is a specialized version of @Controller  It combines the @Controller and @ResponseBody annotations into a single annotation.  On the other hand, @RestController is used for building RESTful APIs, where the controller methods handle requests and directly return the response in a format such as JSON or XML.  @Service  @Service Đánh dấu một Class là tầng Service, phục vụ các logic nghiệp vụ.  @Repository  @Repository Đánh dấu một Class Là tầng Repository, phục vụ truy xuất dữ liệu.  @Autowired (with or without @Qualifier)  @Autowired đánh dấu cho Spring biết rằng sẽ tự động inject bean tương ứng vào vị trí được đánh dấu.  @Primary là annotation đánh dấu trên một Bean, giúp nó luôn được ưu tiên lựa chọn trong trường hợp có nhiều Bean cùng loại trong Context.  @Qualifier xác định tên của một Bean mà bạn muốn chỉ định inject.  public Girl(@Qualifier("naked") Outfit outfit) {  // Spring sẽ inject outfit thông qua Constructor đầu tiên  // Ngoài ra, nó sẽ tìm Bean có @Qualifier("naked") trong context để ịnject  this.outfit = outfit;  }  Sau khi tìm thấy một class đánh dấu @Component. thì quá trình inject Bean xảy ra theo cách như sau:   * Nếu Class không có hàm Constructor hay Setter. Thì sẽ sử dụng Java Reflection để đưa đối tượng vào thuộc tính có đánh dấu @Autowired. * Nếu có hàm Constructor thì sẽ inject Bean vào bởi tham số của hàm * Nếu có hàm Setter thì sẽ inject Bean vào bởi tham số của hàm * The functionality of Dispatcher Servlet, Handler Mapping, Controller, ModelAndView, ViewResolver and View * How the above components interact with each other to process a Request to generate a Response (a work-flow diagram is required) * Explain in depth the most commonly used components:   + **@RequestMapping / @RequestParam**   + **@RequestBody / @ResponseBody** * Explain how the validation input with JSR-303 (Bean Validation) is done with Spring? And how errors can be communicated back to users? * Should we store editing data as instance member of the controller?   How is a RestController different from a Controller?  Return Type Handling: @RestController assumes that the return type of handler methods represents the response body itself, while @Controller typically returns a view name or ModelAndView object.  Annotation Combination: @RestController combines @Controller and @ResponseBody into a single annotation, eliminating the need for explicit @ResponseBody annotations.  View Resolution: @Controller is responsible for handling requests and rendering views, whereas @RestController focuses on processing requests and returning data in a suitable format for consumption by clients.   * When is a bean created and destroyed? * Khi IoC Container (ApplicationContext) tìm thấy một Bean cần quản lý, nó sẽ khởi tạo bằng Constructor * inject dependencies vào Bean bằng Setter, và thực hiện các quá trình cài đặt khác vào Bean như setBeanName, setBeanClassLoader, v.v.. * Hàm đánh dấu @PostConstruct được gọi * Tiền xử lý sau khi @PostConstruct được gọi. * Bean sẵn sàng để hoạt động * Nếu IoC Container không quản lý bean nữa hoặc bị shutdown nó sẽ gọi hàm @PreDestroy trong Bean * Xóa Bean. * Coach checks the exercises. | 2 |
|  | C | Expert validation | Validate the training result with expert | 1 |

## Advanced (optional)

These steps are for experienced trainees and should be used only when the basic plan is completed under the estimated time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Type[[2]](#footnote-2)** | **Task** | **Description** | **Estimated Time (h)** |
|  | R, E | Understand Spring Bean scopes | When you create a bean definition, you create a recipe for creating actual instances of the class defined by that bean definition. The idea that a bean definition is a recipe is important, because it means that, as with a class, you can create many object instances from a single recipe.  Please help to answer the following questions:   * Name all bean scopes are supported out-of-the-box in Spring. * For each listed bean scope, describe their purpose and give one example of their application. * Can you describe in detail how session-scope beans work?   Then, configure a session-scope bean for storing user preference in the existing application. How can we verify that the bean is truely session-scope? | NA |
|  | R, E | Understand Aspect Oriented Programming (AOP) with Spring | AOP complements OOP by providing another way of thinking about program structure. Whereas OO decomposes applications into a hierarchy of objects, AOP decomposes programs into aspects or concerns. This enables the modularization of concerns such as transaction management that would otherwise cut across multiple objects (such concerns are often termed crosscutting concerns).  Please firstly help to answer the following questions:   * What are “aspect”, “join point”, “advice” and “pointcut” in Spring AOP? * How many proxying mechanisms supported in Spring? What are the pros/cons between them? * Give 3 applications of AOP in Spring?   Secondly, configure the existing application to make ProjectService transactional and propose a simple way to verify that the configuration is correct.  Thirdly, configure an annotation-based “around advice” to trace every call to ProjectService by logging a simple message to the console out. | NA |
|  | R, E | Understand Spring MVC’s advanced concepts | Please answer the following questions:   * What is the scope of beans annotated with @Controller in Spring MVC? How can we verify the answer using the developing application? * As you’ve studied about @SessionAttribute before, it is time for a bit more tricky question: Do you think it is thread-safe to use @SessionAttribute in every web application? If it is not, can you find or propose a solution to mitigate or better, completely solve the problem? Then, please implement and verify the solution in the developing application. | NA |

1. R = Reading, E = Exercise, C = Checkpoint, S=Setup, P=Presentation [↑](#footnote-ref-1)
2. R = Reading, E = Exercise, C = Checkpoint, S=Setup, P=Presentation [↑](#footnote-ref-2)