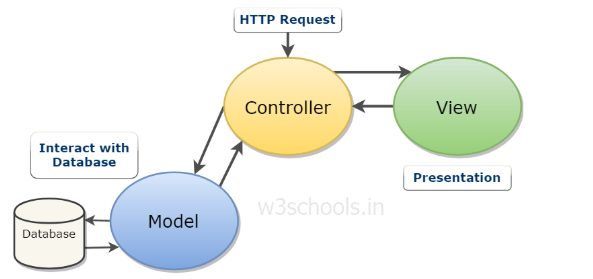
**Spring Framework Some Additional Topics**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**I. Spring MVC Model**

Spring MVC is a model view controller-based web framework under the Spring framework. MVC stands for Model-View-Controller. Let's see picture below:



For more details:

**+ Model**: The model encloses the application related data. Model interacts with the database but doesn’t deal with any logic about how to present the data (business logic).

Example: Files like Entity, DAO to interact with database

**+ View**: The view element is used to present data of the model to the user. This element deals with how to link up with model’s data.

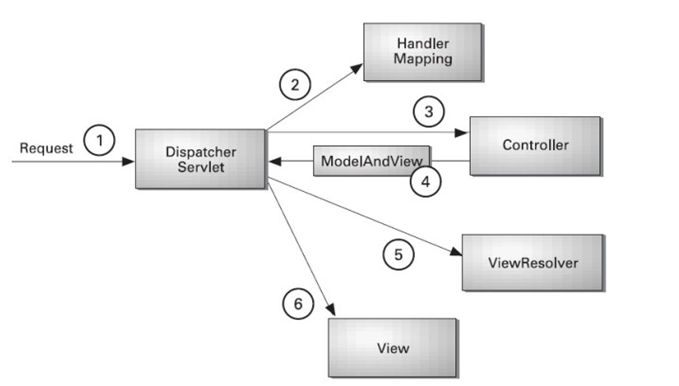
Example: Files like JSP, html

**+ Controller**: The controller is in between the model and the view element. It listens to all the incidents with data processing based on its model from the service layer and actions triggered in the view and performs a response back to the event.

Example: Files like Dispatcher Controller, Handler Mapping, Controller delegate requests.

**II. Program flow in MVC model**

The flow can be described in 6 steps:



1- User send request

2- Dispatcher Servlet (front controller use handler mapping to know what controller deal with this request)

3- Controllers receive requests, call class services (business layer) to resolve.

4- Controller get model from database and send to dispatcher servlet

5- Dispatcher servlet use view resolver to get response.

6- View template, model, view page are built and send to user

**Advantages using Spring MVC:**

-Separate roles: each role can be fulfilled by a specialized object.

-Light-weight: It uses light-weight servlet container

-Powerful Configuration

-Rapid development

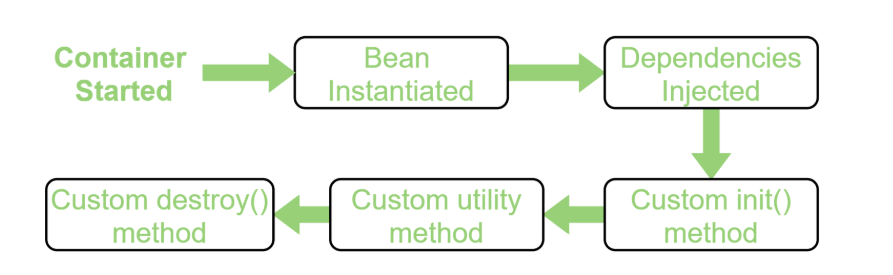
-Reusable business code

-Easy to test

-Flexible mapping

**III. Bean life cycle in Java Spring**

Bean is basically an entity in your application. You can consider it like an object as well as any kind of web application object such as Entity, Configuration component.



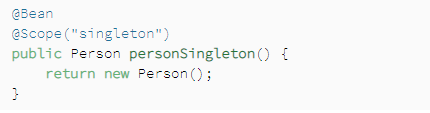
Bean life cycle is managed by spring container. When we run the program, the spring container gets started. After that, the container creates the instance of a bean then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.

Spring bean scope:

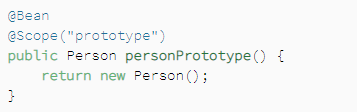
* Singleton (default scope)
* Prototype
* Request
* Session
* Application
* Web socket

**IV. Singleton and prototype scope**

+ **Singleton**: the container creates a single instance of that bean; all requests for that bean name will return the same object.



+ **Prototype**: will return a different instance every time it is requested from the container



**V. BeanFactory and ApplicationContext**

Basically, Bean factory and applicationContext follow abstract design patterns. Normally, we have to create new service object if the client class requires method of that => could lead to tight coupling between each class in the program

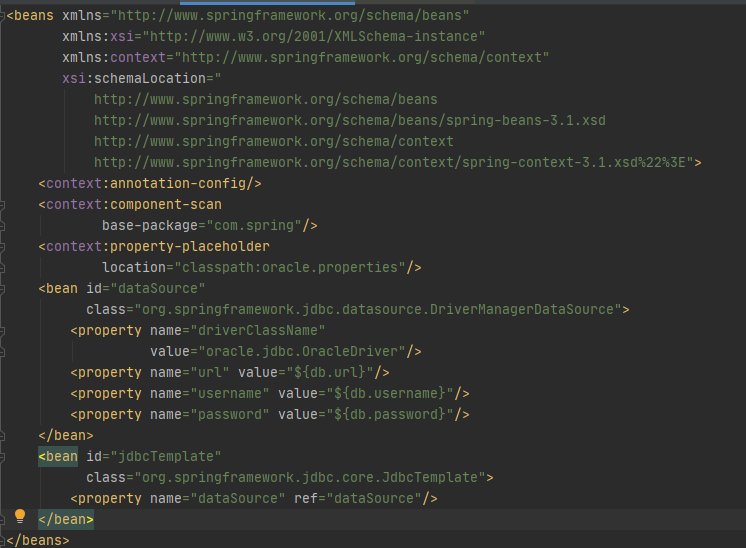
* Both are interface within a spring application that used for providing configuration information to the application
* Application context contains all implementations of bean factory. Bean factory only load the bean when it is required (lazy loading). Application context will load all the beans instantly (eager loading).

Diagram

Description automatically generated

**VI. Config in Spring Framework**

**1. Annotation based configuration**



+ Context:annotation-config: is to scan and active annotation that has been placed inside autowired class such as @autowired, @Qualifire,...

+ Context:component-scan: support annotation config to scan with additional annotations: @Component, @Repository, @Controller, @Bean, @Service…

**2. Java based configuration**

Graphical user interface, website

Description automatically generated

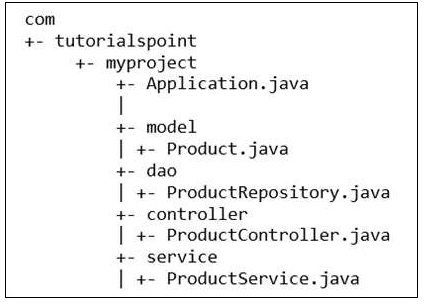
**3. XML-based configuration**

Text

Description automatically generated

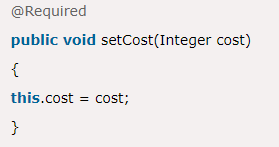
**VII. Typical layout of program**

Example of the typical layout of Spring Boot Application:



**VIII. Introduce some annotation in Spring**

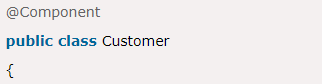
**@Required:** It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property



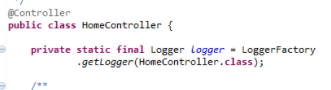
**@Qualifier:** show how to differentiate beans of the same type. Uniquely identifies this bean with its string



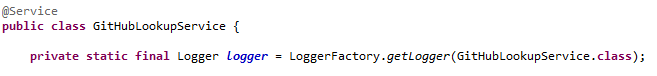
**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. The spring framework pick it up and configure it in the application context as a Spring bean



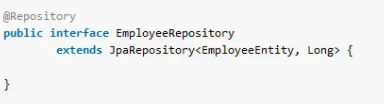
**@Controller:** is class-level annotation. It marks a class as a web request handler. It is mostly used with @RequestMapping annotation



**@Service:** is also used at class level, It tells Spring that class contains business logic.

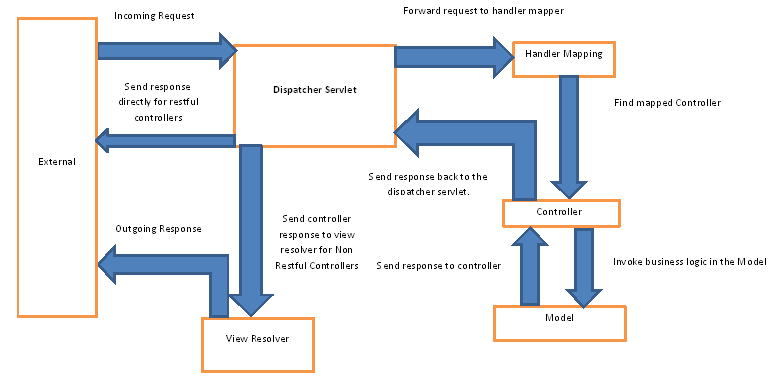


**@Repository:** the repository is a DAOs (Data Access Objects) that access the database directly. The repository does all the operations related to the database.



**IX. Front-controller and flow of a request.**

Front controller is DispatcherServlet class, let’s see picture below again:



A front controller is defined as a controller that handles all requests for a Web Application.

DispatcherServlet servlet is the front controller in Spring MVC that intercepts every request and then dispatches requests to an appropriate controller.

**X. Interceptor**

Interceptor is technique used to perform operations under the following situations:

* preHandle() : Before sending request to the controller
* postHandle() : Before sending the response to the client
* afterCompletion(): Used to perform operations after completing the request and response

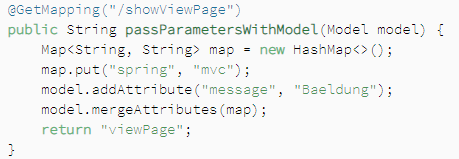
With an interceptor, we could handle pre-phase requests such as retrieving any kind of data related to this request before it is processed or checking your custom authentication. One more ways that we could check the health of the service that needed to be called, etc

**Example:**

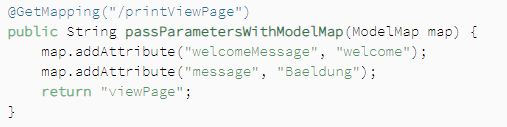


**XI. Model, ModelMap, ModelAndView**

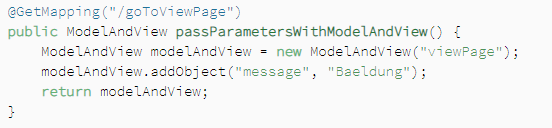
**Model**: the model can supply attributes used for rendering views, we add this data to the object

****

**ModelMap**: just like Model, but ModelMap allows us to parse a collection of values and treat them like Model.



**ModelAndView**: allow us to parse all information required by Spring MVC



We use Interface Model, ModelAndView, ModelMap to pass data from Controller to View for display. Spring allows us to use the Model as a parameter in the Controller's method, so we can easily get and edit the data to pass to the View.

=> Thus they are like a data bridge between the Controller and the View