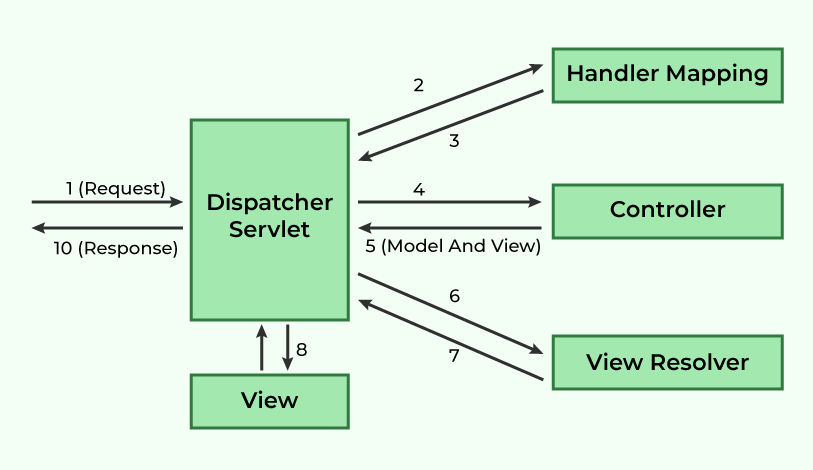
***Spring Web MVC:-***

1. By using Spring Web MVC we can develop web based application as well as Distributed application.
2. It is sub framework of Spring framework.
3. It is built on top of Servlet API



***Flow of Request Handling in Spring MVC***

1. **Client Request → DispatcherServlet**
   * The client (browser or API consumer) sends an **HTTP request**.
   * The request is first received by **DispatcherServlet**, which acts as the front controller.
2. **DispatcherServlet → HandlerMapping**
   * The **DispatcherServlet** consults the **HandlerMapping** to determine the appropriate controller method for handling the request.
3. **HandlerMapping → DispatcherServlet**
   * The **HandlerMapping** returns the matched controller method to the **DispatcherServlet**.
4. **DispatcherServlet → Controller**
   * The **DispatcherServlet** forwards the request to the identified **Controller**.
5. **Controller → DispatcherServlet (Model and View)**
   * The **Controller** processes the request and returns a **ModelAndView** object, which contains:
     + **Model** (data to be displayed)
     + **View Name** (logical name of the view to be rendered)
6. **DispatcherServlet → ViewResolver**
   * The **DispatcherServlet** forwards the **View Name** to the **View Resolver** to resolve the actual view.
7. **ViewResolver → DispatcherServlet (Resolved View)**
   * The **View Resolver** returns the fully resolved view object.
8. **DispatcherServlet → View**
   * The **DispatcherServlet** forwards the model data to the resolved **View**.
9. **View Renders Data**
   * The **View** generates the final HTML response using the provided model data.
10. **Response Sent to Client**

* The final response is sent back to the client.

***Components & Their Duties***

1. **DispatcherServlet (Front Controller)**
   * Central controller that manages the complete request-processing workflow.
   * Delegates tasks to appropriate components.

* It is the class which is used to handle request and response.
* It is also called as Front controller.

1. **HandlerMapping**
   * Maps requests to corresponding controller methods.
   * Uses configurations like **annotation-based (@RequestMapping)** or XML-based mappings.
2. **Controller**
   * Contains business logic.
   * Processes the request and returns a **ModelAndView**.
3. **ModelAndView**
   * Holds both **model data** and **view name**.
   * Helps in rendering dynamic content.
4. **ViewResolver**
   * Resolves logical view names into actual view implementations (e.g., JSP, Thymeleaf, FreeMarker).
5. **View**
   * The final UI template that presents the data to the client.

MVC Components:-

1. Dispatcher Servlet:-

* It is the class which is used to handle request and response.
* It is also called as Front controller.
* **@Controller:-** It is used to represent Java class as MVC Controller / Component.
  + - It is class-level annotation.
    - It consist **@Component** annotation.
    - It is used to return Web page name form controller method.
    - It consist multiple http request binded methods.

For binding http request to particular controller method we can use :-

**@RequestMapping** annotation

* Sending data through request:-

We can send data using URL by using QueryParameter, http request body, http request header,URL.

***QueryParameter:-***

* Data present in QueryParameter in key-value pair.
* Username=xyz🡪 Key , Password= pqr123 🡪Value
* Data starts with **?** and separated with **&** symbol.
* We can take data from QueryParameter by calling respective key.
* So in backend we can take data by using **@RequestParam** annotation.
* Data always present at the end of URL.

1. HandlerMapper / URL Mapper
2. Controller
3. Model and view
4. ViewResolver

By using spring mvc we can develop web based application as well as distributed application

MVC Component:-

1. Dispatcher Servlet: Dispatcher servlet is class which it is handler response it is called as front [controller.](mailto:controller.@controller) @controller it is used represent java class as a mvc component.

It is class level annotation

It consist component annotation

It used to return web page name from controller method

It consist multiple HTTP request bind method For HTTP request to particular controller method we can use @request mapping annotation.

Sending data through request:- We can send data by using URL (query parameter),HTTP request body, HTTP headers, path url

**Query Parameter:- Data present inside key value pair**

**Eg username, password**

Data start with ? and separated with symbol. We can take data from query parameter from calling respected key in backend we can take data by using @request param annotation

Data present at the end parameter

1. URL Handler(Handler Mapper):-
2. Controller
3. Model and view
4. View Resolver

* **Two types of Exception Handling In MVC:**
* Controller & Global Exception are used when there is same type of multiple exception that time we can use this type of exception to reduce broiler plate code.
* **Exception checks flow—try catch-> Controller exception-> Global exception.**

1. **Centralize (Global) Exception Handling :-**

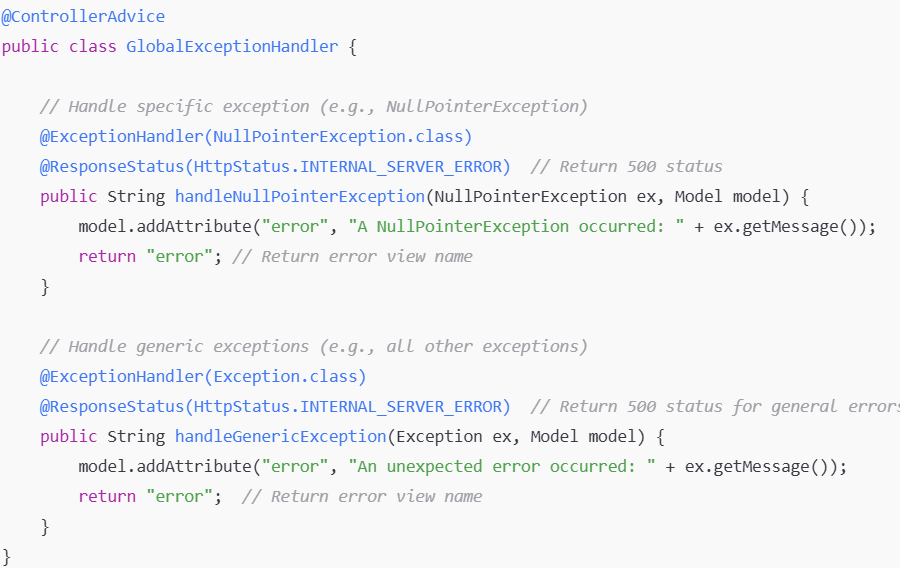
* When there are many controllers classes in that we can’t use only exception handler annotation because it is applicable for only specific controller class.
* If we got many exception in same type in different classes to handle it @controllerAdvice & @ExceptionHandler.

1. **@ControllerAdvice:-**

* It is class level annotation.
* This annotation is used to create a class that globally handles exceptions for controllers.

1. **@ExceptionHandler:-**

* It is method level annotation.
* The @ExceptionHandler annotation is used to define methods that handle specific exceptions (like NullPointerException or Exception).



1. **Controller Based Exception Handling :-**

* When we get same type of exception in many times in that time when it to write try catch block many time so it increase boiler plate code and length of the code to over come this problem we use **@ExceptionHandler**

**Annotation.**

1. **@ExceptionHandler :-**

* It is method level annotation.
* The @ExceptionHandler annotation is used to define methods that handle specific exceptions (like NullPointerException or Exception).
* **Model Interface :-**
* **Model** is typically used in controller methods.
* **Model** is commonly used to hold the data you want to display on the user interface.
* We use the addAttribute method to add an attribute named "greetingMessage" with the value "Hello, Spring MVC!" to the model.
* The data can then be accessed and displayed in the view



* **Uses Of Annotation In Spring MVC :-**

1. **@Controller:-** It is class level annotation.

* This annotation indicating that it can handle HTTP requests.



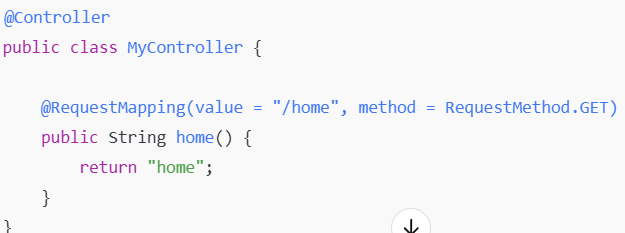
1. **@Autowired :-** It is automatically dependencies injects one class bean into another class.

* This is used for dependency injection in Spring, meaning Spring will automatically provide the required instance of a class without you needing to create it manually.



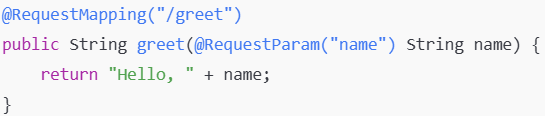
1. **@RequestMapping(value = "") :-**

* It is method level annotation.
* Main purpose of @RequestMapping annotation is Maps HTTP requests to handler methods of MVC controllers.
* This is used to define which URL path a method should handle and the type of HTTP request (GET, POST, etc.).



1. **@RequestParam :-**

* Binds request parameters to method parameters in your controller. It’s used to access data sent by the client, usually via query parameters.



1. **@Service :-** It is Service class level annotation provides some business functionality.

* This is typically used for service layer classes



1. **@Repository :-** It is class level annotation as Data Access Object(DAO).

* This annotation interact with the database.
* It indicates that this class is responsible for data access operations.



1. **@ModelAttribute:-**

* When using **@ModelAttribute** on a method parameter, Spring automatically binds incoming HTTP request parameters (e.g., form data) to a model object.



1. **@Lob (Large Object) :-**

**Purpose**: The @Lob annotation is used to mark a field in an entity class that should be mapped to a large object in the database, such as a BLOB (Binary Large Object) or CLOB (Character Large Object).

**Usage**: This is commonly used for fields that store large data, like images, files, or large text data.

**Example**:-

|  |  |
| --- | --- |
| **1**. **@Lob**  **private byte[] fileData; (BLOB)** | **2.** **@Lob**  **private String description;(CLOB)** |

1. **@CreationTimestamp :-** The @CreationTimestamp annotation is used to automatically populate a field with the current timestamp when an entity is created. It is commonly used for similar fields.

**Usage:** Typically used with java.util.Date or java.time.LocalDateTime fields in an entity.

**Example:- @CreationTimestamp**

**private LocalDate uploadedDate;**

1. **@UpdateTimestamp :-**

**Purpose:** The @UpdateTimestamp annotation is used to automatically update a field with the current timestamp whenever an entity is updated.

**Usage:** This is typically used for fields like lastModified, which track when an entity was last modified.

**Example :- @UpdateTimestamp**

**private LocalDateTime updated\_Date;**

1. **@Column(updatable = false) :-**

The updatable = false attribute means that the column is not updated when the entity is modified.

This is useful for fields that should only be set once and should not be changed after the entity is created.

**Example:- @CreationTimestamp**

**@Column(updatetable = false)**

**private LocalDateTime created\_Date;**

1. **@Resource :-**

The @Resource annotation is a part of Java and is used for dependency injection. It allows Spring to inject resources such as beans, services, or other components into your class.

**Usage**: Typically used to inject a service or DAO into a class.

1. **@Inject :-**

**Purpose**: The @Inject annotation is part of the javax.inject package and is used for dependency injection. It is similar to the Spring @Autowired annotation, but it is a more general-purpose annotation provided by JSR-330. It is used to mark fields, constructors, or methods for dependency injection.

**Usage**: In Spring, it can be used in place of @Autowired to inject dependencies.

**Example**:- @Inject

private MyService myService;