Core Specific- Java Assignment 10

Q1. What is the Spring MVC framework?

Ans:

Spring MVC is a Java framework that is used to develop web applications. It is built on a Model-View-Controller (MVC) pattern and possesses all the basic features of a spring framework, such as Dependency Injection, Inversion of Control.

Q2. What are the benefits of Spring MVC framework over other MVC frameworks?

Ans:

Advantages of Spring MVC Framework

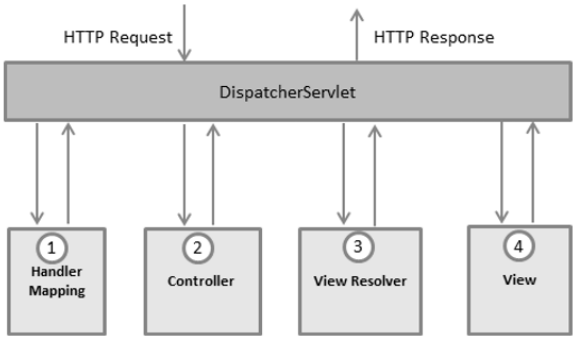
- The container is used for the development and deployment of applications and uses a lightweight servlet.
- It enables rapid and parallel development.
- Development of the application becomes fast.
- Easy for multiple developers to work together.
- Easier to Update the application.
- It is Easier to Debug because we have multiple levels in the application.

Q3. What is DispatcherServlet in Spring MVC? In other words, can you explain the Spring MVC architecture?

Ans:

The *DispatcherServlet* is the front controller in Spring web applications. It's used to create web applications and REST services in Spring MVC. In a traditional Spring web application, this servlet is defined in the *web.xml* file.

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram –



Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet* –

- After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
- The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
- The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
- Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

All the above-mentioned components, i.e., HandlerMapping, Controller, and ViewResolver are parts of *WebApplicationContext* which is an extension of the plain *ApplicationContext* with some extra features necessary for web applications.

Q4. What is a View Resolver pattern and explain its significance in Spring MVC?

Ans:

Spring MVC is a Web MVC Framework for building web applications. In generic all MVC frameworks provide a way of working with views. Spring does that via the **ViewResolvers**, which enables you to render models in the browser without tying the implementation to specific view technology. Now let's understand **ViewResolver** with an example project in STS.
The Spring MVC framework is comprised of the following components:

1. **Model**: A model can be an object or collection of objects which basically contains the data of the application.
2. **View**: A view is used for displaying the information to the user in a specific format. Spring supports various technologies like freemarker, velocity, and thymeleaf.
3. **Controller**: It contains the logical part of the application. @Controller annotation is used to mark that class as controller.
4. **Front Controller**: It remains responsible for managing the flow of the web application. DispatcherServlet acts as a front controller in Spring MVC.

Core Specific- Java Assignment 10

Q5. What are the differences between @RequestParam and @PathVariable annotations?

Ans:

1. Using @RequestParam to get Query parameters

In a Spring MVC application, you can use the @RequestParam annotation to accept query parameters in Controller's handler methods.

For examples, suppose you have a web application that returns details of orders and trades, and you have the following URLs:

http://localhost:8080/eportal/orders?id=1001

To accept the query parameters in the above URLs, you can use the following code in the Spring MVC controller:

```
@RequestMapping("/orders")
public String showOrderDetails(@RequestParam("id") String orderId, Model model){
  model.addAttribute("orderId", orderId);
  return "orderDetails";
}
```

If the name of the query parameter is the same as the name of the variable in the handler's @RequestParam annotated argument then you can simply use @RequestParam without specifying the name of a query parameter, Spring will automatically derive the value.

Also, here is the code to prove the point:
URL: http://localhost:8080/eportal/trades?tradeId=2001

```
@RequestMapping("/trades")
public String showTradeDetails(@RequestParam String tradeId,
                    Model model){
 model.addAttribute("tradeId", tradeId);
 return "tradeDetails";
}
```

You can see that we have just annotated the method parameter tradeId with @RequestParam without

specifying the name of the query parameter because the name of both request parameter and argument name is the same, i.e., "tradeId."

2. Using @PathVariable annotation to extract values from URI

You can use Spring MVC's @PathVaraible annotation to extract any value which is embedded in the URL itself. Spring calls it a URI template, where @PathVariable is used to obtain some placeholders from the URI itself.

If you have worked in RESTful Web services, you might know that the REST URIs contains values, like a REST API to retrieve a book using an ISBN number looks like the following:

URL: http://localhost:8080/book/9783827319333

Now, to extract the value of ISBN number from the URI in your Spring MVC Controller's handler method, you can use @PathVariable annotation as shown in the following code:

```
@RequestMapping(value="/book/{ISBN}", method= RequestMethod.GET)
public String showBookDetails(@PathVariable("ISBN") String id,
                    Model model){
  model.addAttribute("ISBN", id);
  return "bookDetails";
}
```

Similar to @RequestParameter annotation, you also can also omit the value attribute in @PathVariable annotation, if the name of the path variable's placeholder in the @RequestMapping annotation is the same as the variable name in the handler method's @PathVariable annotated parameter.
For example, you can rewrite the above code as shown below:

```
@RequestMapping(value="/book/{ISBN}", method= RequestMethod.GET)
public String showBookDetails(@PathVariable String ISBN,
                    Model model){
  model.addAttribute("ISBN", ISBN);
  return "bookDetails";
}
```

Core Specific- Java Assignment 10

Q6. What is the Model in Spring MVC?

Ans:

Spring Framework provides an Interface called **Model(I)** to work with the data. It defines a placeholder for model attributes and is primarily designed for adding attributes to the model. It is also used to transfer data between the view and controller of the Spring MVC application. Model interface is available in the **org.springframework.ui** package. It acts as a data container that contains the data of the application. That stored data can be of any form such as String, Object, data from the Database, etc.

Q7. What is the role of @ModelAttribute annotation?

Ans:

In Spring MVC, the @ModelAttribute annotation binds a method parameter or method return value to a named model attribute and then exposes it to a web view. It refers to the property of the Model object.

For example, if we have a form with a form backing object that is called "Student" then we can have Spring MVC supply this object to a Controller method by using the @ModelAttribute annotation:

```
@RequestMapping("/home")
public String showHomePage(@ModelAttribute("studentInfo") StudentInfoDTO studentInfoDTO) {

    return "something";

}
```

So let's understand the whole concept of @ModelAttribute Annotation with an interesting example project. Before that, we will suggest you please refer to these articles so that it's going to be very easy for you to understand the concept of @ModelAttribute Annotation through an example project.

- Data Binding in Spring MVC with Example
- Two-Way Data Binding in Spring MVC with Example

Q8. What is the significance of @Repository annotation?

Ans:

@Repository Annotation is a specialization of @Component annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.

Core Specific- Java Assignment 10

Q9. What does REST stand for? and what is RESTful web services?

Ans:

Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work. REST was initially created as a guideline to manage communication on a complex network like the internet.

RESTful web services are generally highly scalable, light, and maintainable and are used to create APIs for web-based applications. It exposes API from an application in a secure and stateless manner to the client. The protocol for REST is HTTP. In this architecture style, clients and servers use a standardized interface and protocol to exchange representation of resources.

Q10. What is differences between RESTful web services and SOAP web services?

Ans:

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.

9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.