SPRING BOOT WEB MVC NOTES

🔰 SECTION 1: MVC BASICS – UNDERSTANDING THE FOUNDATION

🔹 **What is MVC?**

- **MVC** stands for **Model-View-Controller** — a design pattern used to separate application logic.
- **Model** → Represents data. In Java, these are usually **POJOs** (Plain Old Java Objects).
- **View** → Represents UI. Usually implemented using **JSP**, **Thymeleaf**, or **HTML templates**.
- **Controller** → Acts as a bridge. Handles incoming HTTP requests, processes them (maybe using model), and returns a view.

🔹 **Why MVC?**

- Separates concerns: cleaner code, easier to maintain.
- Enables team collaboration (backend & frontend devs can work independently).
- Core principle behind Spring Boot Web Apps.

🔰 SECTION 2: HOW SPRING BOOT USES SERVLETS

🔹 **What is a Servlet?**

- Java class that handles **HTTP requests** and sends **responses**.
- It's the base of all Java web apps.

🔹 **Why use Tomcat?**

- A **Servlet Container** – needed to run servlets.
- Embedded inside Spring Boot (so no need to install separately).

🔹 **Writing a Basic Servlet (for understanding):**

```
public class HelloServlet extends HttpServlet {
    protected void service(HttpServletRequest req, HttpServletResponse res) {
        res.getWriter().println("Hello, User!");
    }
}
```

🔹 **Running Tomcat (Manually):**

```
Tomcat tomcat = new Tomcat();
tomcat.start();
tomcat.getServer().await();
```

✅In Spring Boot, this is abstracted away – you just run your app and it's served via embedded Tomcat.

🔰SECTION 3: SPRING BOOT MVC STRUCTURE (REAL USE CASE)

🔹 **Controller**

- Handles web requests (like `/home`, `/add`)
- Annotated with `@Controller`
- Methods return **view names** (mapped to JSP or HTML files)

```
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home() {
        return "home"; // returns home.jsp
    }
}
```

🔹 **View (JSP Page)**

- Stored in: `src/main/webapp/WEB-INF/views/`
- Used to show the UI with dynamic content

```
<%@ page language="java" %>
<html>
<body>
  <h1>Hello from JSP</h1>
</body>
</html>
```

🔹 **URL Mapping**

- `@RequestMapping("/path")` connects URL to controller method
- Can use `@GetMapping` and `@PostMapping` for more clarity

✅PART A – CALCULATOR PROJECT (ADD TWO NUMBERS)

🔰SECTION 4: CALCULATOR FORM – PRACTICE FLOW

🔹 **index.jsp**

```
<%@ page language="java" %>
<html>
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>Telusko Calculator</h2>
  <form action="add">
    <label>Enter 1st Number:</label>
    <input type="text" name="num1"><br>

    <label>Enter 2nd Number:</label>
    <input type="text" name="num2"><br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

- When submitted, it sends data to `/add` URL with parameters: `num1`, `num2`

🔹 **Link CSS**

```
<link rel="stylesheet" href="style.css">
```

🔰SECTION 5: PROCESSING FORM DATA – SERVLET + SESSION

🔹 **Use `HttpServletRequest` to get form data**

```
@RequestMapping("add")
public String add(HttpServletRequest req, HttpSession session) {
    int num1 = Integer.parseInt(req.getParameter("num1"));
    int num2 = Integer.parseInt(req.getParameter("num2"));
    int result = num1 + num2;
```

```
    session.setAttribute("result", result); // Stores result value in session
    return "result.jsp";
}
```

- `req.getParameter()` reads value from the form.
- `session.setAttribute()` stores result temporarily.

🔹 **What is HttpSession?**

- `HttpSession` is an interface provided by Servlet API.
- It is used to **store data specific to a user across multiple requests**.
- Think of it as a temporary storage space for a particular user's browser session.
- Stored on server-side, each user gets a unique session ID (via a cookie).

✅Use Cases:

- Store login status
- Store intermediate calculation results
- Store data between pages

🔹 result.jsp – Show Result using session

Only the key line to display result:

```
<h2>Result is: <%= session.getAttribute("result") %></h2>
```

✅Alternative (optional): You can use JSTL:

```
<h2>Result is: ${result}</h2>
```

🔰SECTION 6: SPRING WAY – MATCHING FORM DATA TO METHOD ARGUMENTS

🔹 **Spring allows automatic binding of form fields to method parameters using annotations or by matching names.**

```
@RequestMapping("add")
public String add(@RequestParam("num1") int num, int num2, HttpSession session)
{
    int result = num + num2;
    session.setAttribute("result", result);
```

```
    return "result.jsp";
}
```

- If parameter names match variable names, you can skip the annotation:

```
public String add(int num1, int num2, HttpSession session)
```

✅This helps avoid manual parsing like `request.getParameter()` and keeps the controller cleaner.

🔰SECTION 7: USING ModelAndView INSTEAD OF HttpSession

🔹 **Why move away from `HttpSession`?**

- `HttpSession` is useful but stores data tied to user sessions.
- It persists data longer than necessary (until session expires or is invalidated).
- Can lead to memory issues if not managed properly in large-scale apps.

🔹 **What is ModelAndView in Spring?**

- `ModelAndView` is a class that lets you pass both **data (model)** and the **view name** in a single return object.
- Instead of using `Model model` or `HttpSession`, you use `ModelAndView` to attach values and view in one go.

🔹 **Example – Using `ModelAndView` in method parameters:**

```
@RequestMapping("add")
public String add(@RequestParam("num1") int num, int num2, ModelAndView mv) {
    int result = num + num2;
    mv.setViewName("result.jsp");
    mv.addObject("result", result);
    return mv;
}
```

- `setViewName()` tells Spring which page to render.
- `addObject()` adds data you want to show in that view (like the sum result).

✅In this approach, Spring injects the `ModelAndView` object for you — no need to create it manually.

🔹 **Benefits:**

- Clean: handles multiple fields easily
- Object-Oriented: promotes better data handling
- Stateless: avoids session usage

• Reusable: pass entire objects between controller and view

@RequestMapping("add") public ModelAndView add(@RequestParam("num1") int num, int num2) { int result = num + num2;

```
ModelAndView mv = new ModelAndView();
mv.setViewName("result.jsp");
mv.addObject("result", result);

return mv;
```

}

```
- `setViewName()` tells Spring which page to render.
- `addObject()` adds data you want to show in that view (like the sum result).
}
```

• Now `result.jsp` can access `${result}` directly — no session needed.

🔹 **Benefits:**

• Clean: handles multiple fields easily
• Object-Oriented: promotes better data handling
• Stateless: avoids session usage
• Reusable: pass entire objects between controller and view

✅PART B – STUDENT FORM PROJECT (USING @ModelAttribute WITHOUT MODEL OR MODELANDVIEW)

🔰SECTION 8: HANDLING STUDENT FORM WITH @ModelAttribute

🔹 **Project Summary:**

• A form is used to submit a **Student** object (e.g., name, age).
• The controller uses `@ModelAttribute Student student` to receive form data.
• You're **not using `or \*\*` – and that's perfectly fine for this case.

🔹 **Why it Works Without `Model model`:**

- Spring automatically adds the form-bound object (`Student`) to the model.
- It uses the class name (converted to lowercase) as the key — in this case, `student`.
- You're returning the view name (`"result"`) directly, so no need to configure `ModelAndView`.

```
@PostMapping("/result")
public String result(@ModelAttribute Student student) {
    return "result";
}
```

🔹 **In result.jsp**

You can access the student object directly using EL (Expression Language):

```
<p>Name: ${student.name}</p>
<p>Age: ${student.age}</p>
```

✅Simple and clean – no extra object handling required.

🔰SECTION 9: When Do You Need `Model model`?

🔹 **Use `Model model` When:**

- You want to manually add extra data to the view.
- Example: `model.addAttribute("message", "Submitted successfully")`
- You want to change the name of the object sent to the view.
- You’re passing more than one object to the view.

🔹 **Skip `Model model` When:**

- You're just using `@ModelAttribute` to bind form data to a single object.
- You’re okay with the default variable name (i.e., class name in lowercase) being used in the view.
- You're not adding extra data manually.

✅Summary: Use `Model model` if you need more control over data being sent to the view. Else, Spring can handle it for you.

✔️Ready for the next part!