# ✅ Notes: Request Path Rules for Handler Methods in Spring MVC

### ✅ 1️⃣ The Request Path Must Start with “/”

* The value of @RequestMapping(value = ...) should always start with a slash /.

✅ Correct:

@RequestMapping(value = "/report")

public String showReport(Map<String, Object> map) { ... }

❌ Incorrect:

@RequestMapping(value = "report") // Missing leading slash → May not work

### ✅ 2️⃣ The Request Path Is Case Sensitive

* The URL path is treated exactly as written, including uppercase and lowercase letters.

✅ Example Controller:

@Controller

public class TestController {

@RequestMapping(value = "/REPORT")

public String showReport(Map<String, Object> map) {

return "displayReport";

}

@RequestMapping(value = "/report")

public String showReport1(Map<String, Object> map) {

return "displayReport1";

}

}

#### ✅ Example URL Mapping

| **Request URL** | **Handler Method Called** |
| --- | --- |
| <http://localhost:9999/ThirdApp/REPORT> | showReport() |
| <http://localhost:9999/ThirdApp/report> | showReport1() |

⚠ Note:  
/REPORT ≠ /report → Different methods will be called.

### ✅ Why These Rules Matter

| **Rule** | **Reason** |
| --- | --- |
| Leading Slash ("/") | Ensures proper mapping and avoids mistakes. |
| Case Sensitivity | Avoids ambiguity; keeps URL mapping predictable. |

### ✅ Best Practice Tip

👉 Always write handler methods like this in real-time projects:

@RequestMapping(value = "/example")

public String exampleHandler(Model model) {

model.addAttribute("key", "value");

return "exampleView"; // Clear and explicit view

}

### 🔹 From Spring 4.3 onward, we have **shortcut annotations** for different types of HTTP requests:

#### ✅ @GetMapping

* Used when we want to handle a **GET request**.
* GET is used to **fetch data** (like loading a webpage or getting details).
* Example:

@GetMapping("/students")

public List<Student> getAllStudents() {

return studentService.getStudents();

}

* This method will run when the browser or client makes a GET request to /students.

#### ✅ @PostMapping

* Used when we want to handle a **POST request**.
* POST is used to **send data to the server** (like submitting a form or saving new data).
* Example:

@PostMapping("/students")

public String saveStudent(@RequestBody Student student) {

studentService.save(student);

return "Student saved";

}

* This method runs when data is **sent** (posted) to /students.

# ✅ Notes: Handling Same Request Path with Different Request Methods (GET / POST)

### ✅ 1️⃣ Scenario:

👉 Same URL path (/report) but different HTTP methods →

* One method handles **GET requests**.
* Another method handles **POST requests**.

### ✅ Example Controller Class:

@Controller

public class TestController {

@GetMapping("/") // Handles GET requests for "/"

public String showHome() {

System.out.println("TestController.showHome()");

return "home"; // Returns home.jsp

}

@RequestMapping(value = "/report", method = RequestMethod.GET)

public String showReport(Map<String, Object> map) {

System.out.println("TestController.showReport(-)");

return "display"; // Returns display.jsp

}

@RequestMapping(value = "/report", method = RequestMethod.POST)

public String showReport1(Map<String, Object> map) {

System.out.println("TestController.showReport1(-)");

return "display1"; // Returns display1.jsp

}

}

### ✅ 2️⃣ How It Works in Practice

#### home.jsp:

<form action="report" method="POST"> <!-- POST Request -->

<input type="submit" value="Send"/>

</form>

<br>

<a href="report">link1</a> <!-- GET Request -->

#### ✅ Example Requests and Output

| **Request Type** | **URL** | **Controller Method Called** | **Final JSP Rendered** |
| --- | --- | --- | --- |
| GET | /report | showReport() | display.jsp |
| POST | /report | showReport1() | display1.jsp |

### ✅ 3️⃣ Cleaner Way: Use @GetMapping and @PostMapping

👉 Instead of writing this:

@RequestMapping(value = "/report", method = RequestMethod.GET)

✔ Use this (simpler):

@GetMapping("/report")

public String showReport(Model model) { ... }

✔ And this:

@PostMapping("/report")

public String showReport1(Model model) { ... }

✅ Why use @GetMapping and @PostMapping:

* Cleaner and easier to read.
* Avoids verbosity of specifying method in @RequestMapping.

### ✅ 4️⃣ Simple Summary Table

| **Annotation** | **Purpose** |
| --- | --- |
| @GetMapping | Maps HTTP GET request to handler method |
| @PostMapping | Maps HTTP POST request to handler method |
| @RequestMapping(value=..., method=GET/POST) | Works, but more verbose and old style |

### ✅ Final Simple Thought

👉 In real-world projects:

* Prefer using @GetMapping("/report") and @PostMapping("/report") for simplicity and clarity.

⚠ Do not use just @RequestMapping with method attribute unless needed for legacy support.

# ✅ Notes: Important Handler Method Rules in Spring MVC (Industry Recommended)

### ✅ d. Default Request Path Is “/” if Not Specified

👉 These two handler methods are exactly the same and both are correct:

@GetMapping("/")

public String showHome(Model model) {

System.out.println("TestController.showHome()");

return "home"; // Returns home.jsp

}

@GetMapping // No path specified → Defaults to "/"

public String showHome(Model model) {

System.out.println("TestController.showHome()");

return "home"; // Returns home.jsp

}

✅ Spring treats @GetMapping without a path as equivalent to @GetMapping("/").  
⚡️ This simplifies defining the home page handler.

### ✅ e. One Handler Method Can Handle Multiple Request Paths

👉 Recommended Way (Industry Practice):

@GetMapping(value = {"/report", "/report1", "/report2"})

public String showReport(Model model) {

System.out.println("TestController.showReport()");

return "display"; // Returns display.jsp

}

#### ✅ Example Request URLs

| **Request URL** | **Handler Method Called** |
| --- | --- |
| <http://localhost:9999/ThirdApp/report> | showReport() |
| <http://localhost:9999/ThirdApp/report1> | showReport() |
| <http://localhost:9999/ThirdApp/report2> | showReport() |

👉 All three URLs will call the same handler method → showReport().

### ✅ Why This Way Is Better

| **Feature** | **Benefit** |
| --- | --- |
| Use of Model Parameter | Cleaner, semantically correct, preferred by industry |
| Explicit Logical View Name (LVN) | Clear and predictable |
| Multiple Request Paths in One Method | Avoids code duplication, makes the code DRY |

### ✅ Best Practice Reminder

✔ Always prefer the following format in real projects:

@GetMapping("/example")

public String exampleHandler(Model model) {

model.addAttribute("key", "value");

return "exampleView"; // Explicit Logical View Name

}

⚠ Avoid using Map<String, Object> unless necessary, and never return a Model object as the return type.

### ✅ Final Simple Thought

👉 Always write controller methods in the industry-recommended way:

* Use Model parameter to add data.
* Return **String Logical View Name** explicitly.
* Use @GetMapping, @PostMapping, etc., instead of verbose @RequestMapping

# ✅ Notes: Rule – Request Path + HTTP Method Should Be Unique in Controller Class

### ✅ f. Request Mapping Must Be Unique in the Same Controller Class

👉 It is a **bad practice** to have multiple handler methods in the same controller class with the same combination of:

* URL Path (request pattern)
* HTTP Method (GET/POST)

#### ⚠ Incorrect Example (Bad Practice – Causes Ambiguity)

@GetMapping(value = {"/report"})

public String showReport(Model model) {

System.out.println("TestController.showReport()");

return "display";

}

@GetMapping(value = {"/report"})

public String showReport1(Model model) {

System.out.println("TestController.showReport1()");

return "display";

}

❌ Problem:  
Spring will throw an exception at startup —  
Ambiguous mapping. Cannot map multiple methods to same path + HTTP method combination.

#### ✅ Correct Approach – Use Unique URL or HTTP Method

Example 1 – Different URL Paths:

@GetMapping("/report")

public String showReport(Model model) {

System.out.println("TestController.showReport()");

return "display";

}

@GetMapping("/report1")

public String showReport1(Model model) {

System.out.println("TestController.showReport1()");

return "display1";

}

**✅ Notes: Example 2 – Same URL Path but Different HTTP Methods**

**✅ Why This Is Useful**

👉 Sometimes the same URL (request path) is used for different purposes based on the HTTP method:

* **GET** → Used to display data or show a form.
* **POST** → Used to submit form data or perform an action.

**✅ Correct Example in Industry Practice**

@GetMapping("/report")

public String showReport(Model model) {

System.out.println("TestController.showReport()");

return "display"; // Show report.jsp page

}

@PostMapping("/report")

public String submitReport(Model model) {

System.out.println("TestController.submitReport()");

return "displaySubmit"; // Show displaySubmit.jsp page after form submit

}

**✅ Simple Flow Example**

| **Step** | **Action** | **Outcome** |
| --- | --- | --- |
| 1 | User opens URL in browser: <http://localhost:9999/ThirdApp/report> (GET request) | showReport() executes → Displays report.jsp |
| 2 | User submits form with action="/report" and method="POST" | submitReport() executes → Displays displaySubmit.jsp |

**✅ Why It’s a Good Practice**

* ✅ Separates responsibilities clearly:
  + GET → For showing the form or data.
  + POST → For processing data submitted by user.
* ✅ Prevents mixing read and write operations in one method.

**✅ Important Reminder**

⚡ Do NOT use the same URL + same HTTP method twice in a controller class → Causes ambiguity error.

✔ Instead:

* Either use different URL paths.
* Or use different HTTP methods (GET/POST) for the same URL.

#### ✅ Special Case – Default Path “/” Should Also Be Unique

⚠ Incorrect Example (Bad Practice):

@GetMapping

public String showHome(Model model) {

System.out.println("TestController.showHome()");

return "home";

}

@GetMapping

public String showHome1(Model model) {

System.out.println("TestController.showHome1()");

return "home";

}

❌ Problem:  
Both methods map to the same path “/” and same HTTP method → Causes startup failure due to ambiguity.

✅ Correct Example:

@GetMapping("/")

public String showHome(Model model) {

System.out.println("TestController.showHome()");

return "home";

}

⚡ One handler method for “/” + GET → Always unique.

### ✅ Simple Summary Table

| **Rule** | **Why Important** |
| --- | --- |
| Unique (Path + HTTP Method) | Prevents ambiguity errors at startup |
| Different URL or HTTP Method | Clear responsibility for each handler |
| Default “/” Mapping | Should be defined only once per controller |

# ✅ Notes: Advanced Spring MVC Handler Method Rules (Industry Recommended)

### ✅ g. At Most Two Handler Methods Without Request Path in Same Controller Class

👉 Rule:

* In a single controller class, **maximum two handler methods can exist without request path**:
  + One for **GET requests**.
  + One for **POST requests**.

#### ✅ Correct Example:

@PostMapping

public String showHomePost(Model model) {

System.out.println("TestController.showHome()");

return "home"; // For POST request to "/"

}

@GetMapping

public String showHomeGet(Model model) {

System.out.println("TestController.showHome1()");

return "home"; // For GET request to "/"

}

⚡ Spring treats these as:

| **URL Path** | **Request Mode** | **Handler Method** |
| --- | --- | --- |
| “/” | GET | showHomeGet() |
| “/” | POST | showHomePost() |

### ✅ 1️⃣ Same Request Path in Multiple Controller Classes → Use Global Request Path

#### ⚠ Problem Example (Bad Practice):

@Controller

public class DemoController {

@GetMapping("/report")

public String generateReport(Model model) {

return "display";

}

}

@Controller

public class TestController {

@GetMapping("/report")

public String generateReport(Model model) {

return "display";

}

}

❌ Why Bad Practice:

* Both handler methods are mapped to the same request path (/report) with same HTTP method (GET).
* Spring cannot decide which method to call → Application startup fails with an ambiguity error.

#### ✅ Industry Recommended Solution – Add Global Request Path at Controller Level

@Controller

@RequestMapping("/demo") // Global path for DemoController

public class DemoController {

@GetMapping("/report")

public String generateReport(Model model) {

return "display";

}

}

@Controller

@RequestMapping("/test") // Global path for TestController

public class TestController {

@GetMapping("/report")

public String generateReport(Model model) {

return "display";

}

}

#### ✅ Final URL Mapping

| **Request URL** | **Handler Method Called** |
| --- | --- |
| <http://localhost:9999/ThirdApp/demo/report> | DemoController.generateReport() |
| <http://localhost:9999/ThirdApp/test/report> | TestController.generateReport() |

👉 Why This Is Good Practice:

* Avoids ambiguity.
* Makes URL structure meaningful and organized.
* Easier to maintain in large projects.