```
eg#1.
@Controller
@RequestMapping("/")
public class LoginController{

	@GetMapping(value="")
	public String <methodName>(Map<String,Object> model,....){

		return "";
	}

	@PostMapping(value="")
	public String <methodName>(Map<String,Object> model,....){

		return "";
	}
}
```

The possible parameters for Handler method arguments are

a. javax.servlet.ServletRequest
b. javax.servlet.ServletResponse
c. @PathVariable
d. @RequestParam
e. @RequestHeader
f. @RequestAttribute
g. @ModelAttribute
h. Errors,BindingResult
i. @SessionAttribute

The possible return types of Handler methods

a. String
b. View
c. Model
d. @ModelAttribute
e. ModelAndView
f. void

DataBinding and DataRendering
==========

DataBinding :: The process of writing input values(form data/request parameter data) to java class object is called "DataBinding".

DataRendering :: The process of giving controller generated/gathered results/outputs after executing buisness logic to viewcomponents through SharedMemory (BindingAwareModelMap object) is called "DataRendering".

Note:
DataBinding => Binding the data from view component to controller in the form of java class object called Model class object.
DataRendering=> Passing data/results/outputs from Controller to view Component as ModelAttributes through SharedMemory called BindingAwareModelMap object.

Passing different types of results/ouputs/data in DataRendering

a. Passing simplevalues
b. Passing array/collections

c. Passing collection of Model/Buisness Object
d. Passing single Object of Model/Buisness Object/Entity class.

Note: Model(I) comes from SpringFramework, where as Model is Bo/Entity class.

Reading Simple values
==========
```
@Controller
public class TestController {

	@GetMapping("/report")
	public String showReport(Map<String,Object> map) {
		System.out.println("TestController.showReport()");
		map.put("name", "kohli");
		map.put("age", 35);
		map.put("address", "RCB");
		return "show_report";
	}
}
```

show_report.jsp
==========
```
<h1 style="color:red; text-align: center;"> READING SIMPLE VALUES</h1>
<b>NAME IS :: ${name}</b><br/>
<b>AGE IS :: ${age}</b><br/>
<b>ADDR IS :: ${address}</b><br/>
```

Passing arrays and collection values from controller class to view component in DataRendering process
==========
```
package in.ineuron.controller;

import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class TestController {

	@GetMapping("/report")
	public String showReport(Map<String, Object> map) {
		System.out.println("TestController.showReport()");

		String nickNames[] = new String[] { "sachin", "saurav", "dravid", "kohli" };
		Set<Long> mobilePhonesSet = new HashSet<Long>();
		mobilePhonesSet.add(9999999L);
		mobilePhonesSet.add(8888888L);
		List<String> coursesList = List.of("java", "spring", "spring boot", "hibernate");
		Map<String, Long> idsMap = Map.of("aadhar", 4543535L, "voterId", 53543543534L, "panNo", 4545355454L);
```

```
            //Creating a ModelAttributes
            map.put("nickNames", nickNames);
            map.put("phonesInfo", mobilePhonesSet);
            map.put("coursesInfo", coursesList);
            map.put("idsInfo", idsMap);

            return "show_report";
      }
}
```

show_report.jsp
==========

```
<b>Arrays Data</b>
<br />
      <c:forEach var="name" items="${nickNames}">
            ${name }<br />
      </c:forEach>
<hr />

<b>List Data</b>
<br />
      <c:forEach var="course" items="${coursesInfo}">
            ${course }<br />
      </c:forEach>
<hr />

<b> Phone number (set)::</b>
      <br>
      <c:forEach var="phone" items="${phonesInfo}">
                  ${phone} <br>
      </c:forEach>
<hr/>

<b> ids Info (Map)::</b>
      <br>
      <c:forEach var="id" items="${idsInfo}">
                  ${id.key} ===> ${id.value} <br>
      </c:forEach>
```

Passing List of Model class Objects to view component from Controller Using DataRendering Process
==========

Employee.java
==========

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
      private Integer eno;
      private String ename;
      private String desg;
      private Double salary;

}
```

TestController.java
==========

```
@Controller
public class TestController {

	@GetMapping("/report")
	public String showReport(Map<String, Object> map) {
		System.out.println("TestController.showReport()");
		List<Employee> empsList = List.of(new Employee(10, "sachin", "batsman",
90000.0),
				new Employee(7, "dhoni", "keeper", 190000.0), new
Employee(18, "kohli", "captain", 180000.0));
		map.put("empsInfo", empsList);
		return "show_report";
	}
}

show_report.jsp
===============
<table border="1" align="center">
	<tr>
		<th>eno</th>
		<th>ename</th>
		<th>desg</th>
		<th>salary</th>
	</tr>
	<c:forEach var="emp" items="${empsInfo}">
		<tr>
			<td>${emp.eno}</td>
			<td>${emp.ename}</td>
			<td>${emp.desg}</td>
			<td>${emp.salary}</td>
		</tr>
	</c:forEach>
</table>
```

Passing single model class object as a model attribute from controller to view component in data rendering process

```
================================================================================
================================
Employee.java
=============
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
	private Integer eno;
	private String ename;
	private String desg;
	private Double salary;

}

TestController.java
===================
@Controller
public class TestController {
	@GetMapping("/report")
	public String showReport(Map<String, Object> map) {
		System.out.println("TestController.showReport()");
```

```
            Employee employee = new Employee(10, "sachin", "batsman", 90000.0);
            map.put("emp", employee);
            return "show_report";
      }
}
```

show_report.jsp
==========

```
<table border="1" align="center">
      <tr>
            <th>eno</th>
            <th>ename</th>
            <th>desg</th>
            <th>salary</th>
      </tr>
      <tr>
            <td>${emp.eno}</td>
            <td>${emp.ename}</td>
            <td>${emp.desg}</td>
            <td>${emp.salary}</td>
      </tr>
</table>
```

Conclusion of DataRendering in SpringBootMVC App
==========

=> It is the process of passing data from controller class handler methods to view components through DispatcherServlet using Sharedmemory called "BindingAwareModelMap".

DataBinding
==========

=> It is the process of giving the view comps supplied input values to handler methods of controller class

View to controller data passing :: Data Binding
Controller to View data passing :: Data Rendering

Binding can be done in 2 ways
==========

a. Binding form data to handler method of controller class as the Model/Command class Object using "@ModelAttribute".It is also called as "FormBinding/RequestWrapping".

b. Binding hyperlink generated additional request param values to handler method of Handler class using "@RequestParam".It is also called as "RequestParam Binding".

a. Binding form data to handler method of controller class as the Model object using "@ModelAttribute".

Model class/JavaBean class
==========

=> The java bean class whose object holds form component values of form page is called "Model class".

For form binding/data binding we need to follow the operations

=> Count form components in form page and take same no of properties in Model class

=> Make sure that form component names and Model class attributes names should match.

 => Add getter and setter methods for the Properties of the Model class
 => Take Handler method in Handler class having @ModelAttribute<model class type> parameter.

register.jsp
============
```
<form  method="POST">
     <table align="center">
        <tr><td>Employee number::  </td> <td><input type="text" name="eno">
</td></tr>
        <tr><td>Employee name::  </td> <td><input type="text" name="ename">
</td></tr>
        <tr><td>Employee address::  </td> <td><input type="text" name="eadd">
</td></tr>
        <tr><td>Employee  salary::  </td> <td><input type="text" name="salary">
</td></tr>
        <tr><td colspan="2"><input type="submit"  value="register"> </td> </tr>
     </table>
 </form>
```

Employee.java
=============
```
@Data
public class Employee {
      private  int eno;
      private  String ename;
      private  String eadd="hyd";
      private  float salary;

      public Employee() {
            System.out.println("Employee:: 0-param constructor");
      }

}
@Controller
public class EmployeeController {

      @PostMapping("/emp_register")
      public  String  regiserEmployee(Map<String,Object> map,@ModelAttribute("emp")
Employee emp) {
        //read and use form data from model class object or send to service class
            System.out.println(emp);
            return "result";
      }
}
```

Internal Operations of FormBinding
==================================
1. End user fills up the formpage and submit the request
2. Dispatcher servlet traps and take the request
3. Dispatcher Servlet gets handler method signature through "RequestMappingInfo" component.
4. Dispatcher Servlet noticies @ModelAttribute("emp") Employee emp type parameter and it understands to perform Databinding/FormBinding
   by taking Employee class as the Model class.
5. Creates a Model class Object having the name given in the @ModelAttribute("emp") as the object name
                  Employee emp = new Employee()
            Note: If @ModelAttribute is taken without param then it takes the Model

class name like Employee and create the Object having
class name as the object name with the first letter lowercase.
eg: Employee employee = new Employee();
6. Reads the data using request.getParameter("") and it performs the necessary conversion according to the Model class properties using
PropertyEditors
7. Writes the received and converted form data to Model class object using setter methods
8. Dispatcher servlet creates another necessary objects like BindingAwareModelMap Object and calls handler method having those objects
and Model class object(emp).

Note:
Generally we take two handler methods in Controller class with respect to form page operation
a. First Handler method in GET Mode to Lanuch Form page.
b. Second Handler method in POST mode to process from page submission request.
For both the handler methods we can take same request path with different request mode(recomended)
If the above request handler methods are having two different requests path then the request mode are your choice.

Taking same request path for both handler methods[Form launching and submit request processing form page]
=> Taking action attribute in <form ..> becomes optional.
=> When form page is launched we take initial data from Model class object properties and We can display in form page components as
initial values.[It is possible if form page is designed using spring suplied jsp tags]

EmployeeController.java
==========
```
@Controller
public class EmployeeController {

	@GetMapping("/")
	public String showHome() {
		return "home";
	}

	@GetMapping("/emp_register")
	public String showForm() {
		return "register";
	}

	@PostMapping("/emp_register")
	public String regiserEmployee(Map<String, Object> map, @ModelAttribute Employee emp) {
		// read and use form data from model class object or send to service class
		System.out.println(emp);
		return "result";
	}

}
```

Employee.java
==========
```
@Data
```

```
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
	private Integer eno;
	private String ename;
	private String eadd;
	private Double salary;

}
```

```
home.jsp
========
<h1 style="text-align: center">
	<a href="emp_register">Register Employee</a>
</h1>
```

```
register.jsp
============
<form   method="POST">
     <table align="center">
          <tr><td>Employee number::  </td> <td><input type="text" name="eno">
</td></tr>
          <tr><td>Employee name::  </td> <td><input type="text" name="ename">
</td></tr>
          <tr><td>Employee address::  </td> <td><input type="text" name="eadd">
</td></tr>
          <tr><td>Employee  salary::  </td> <td><input type="text" name="salary">
</td></tr>
          <tr><td colspan="2"><input type="submit"  value="register"> </td> </tr>
     </table>
 </form>
</body>
</html>
```

```
result.jsp
==========
<center>
          <h1 style="color: red; text-align: center">Result page</h1>
          <b>form data :: ${employee} </b> <br> <a href="./">home</a>
</center>
```

```
If we use HTML Forms, then by default we will have support only for
"OneWayBinding".[Form to Model class Object]
To get the support of TwoWayBinding then we need to go for "SpringMVC jsp
taglibraries".[Form->Model, Model->Form]
```

```
SpringMVC taglibrary
====================
1. Generic Taglibrary
          <%@ taglib prefix="form" uri="http://www.springframework.org/tags" %>
2. Form tag library
          <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>

          refer:: BootMVCPro6-DataBindingAppUsingSpringTagLibrary
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1" isELIgnored="false"%>
```

Home.jsp
========

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Home Page</title>
</head>
<body>
	<center>
		<h1 style="color: red; text-align: center;">Employee Registration
			Page</h1>
		<form:form method ="POST" modelAttribute="emp">
			<table>
				<tr>
					<th>ENO</th>
					<td><form:input path='eno' /></td>
				</tr>
				<tr>
					<th>ENAME</th>
					<td><form:input path='ename' /></td>
				</tr>
				<tr>
					<th>EDESG</th>
					<td><form:input path='edesg' /></td>
				</tr>
				<tr>
					<th>SALARY</th>
					<td><form:input path='salary' /></td>
				</tr>
				<tr>
					<td></td>
					<td><input type='submit' value='register'></td>
				</tr>
			</table>
		</form:form>
	</center>
</body>
</html>
```

Employee.java
=============

```
@Data
public class Employee {
	private Integer eno;
	private String ename="sachin";
	private String edesg;
	private Double salary;
}
```

EmloyeeController.java
======================

```
@Controller
public class EmployeeController {

	@GetMapping("/register")
	public String showForm(@ModelAttribute("emp") Employee employee) {
```

```
		return "home";
	}

	@PostMapping("/register")
	public String registerEmployee(Map<String, Object> model,
@ModelAttribute("emp") Employee employee) {
		System.out.println("EmployeeController.registerEmployee()");
		System.out.println(employee);
		model.put("employee", employee);
		return "result";
	}
}
```

result.jsp

```
==========
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1" isELIgnored="false"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<center>
		<h1 style='color:red; text-align: center;'>EMPLOYEE DATA</h1>
		<table border='1'>
			<tr>
				<th>ENO</th>
				<td>${employee.eno}</td>
			</tr>
			<tr>
				<th>ENAME</th>
				<td>${employee.ename}</td>
			</tr>
			<tr>
				<th>EDESG</th>
				<td>${employee.edesg}</td>
			</tr>
			<tr>
				<th>SALARY</th>
				<td>${employee.salary}</td>
			</tr>
		</table>
	</center>
</body>
</html>
```

What is the difference b/w HTML Tags vs SpringMVC jsp tags?

HTML tags

```
=========
a. Supports one way binding(Form -> Model)
b. Given by W3C
c. Defult request method type is  "GET"
d. These tags are executed by HTML interpreter
e. Not recomended to use in SpringMVC
```

SpringMVC jsp tags

```
=================
a. Supports two way binding(Form-> Model,Model-> Form)
b. Given by Pivotal team
c. Default request method type is "POST"
d. JSP tags will be converted to HTML tags having the values collected from Model
object as th initial values
e. Recomended to use in SpringMVC

DataBinding using @RequestParam
==============================
 => The request param in the query string either directly or by using hyperlink can
be bound to handler method params of controller class
    by using the support of "@RequestParam" Annotation.



case1::http://localhost:9999/DataBindingApp/data?sno=10

@GetMapping("/data")
public String bindData(Map<String, Object> model, @RequestParam Integer
sno,@RequestParam String sname) {
            System.out.println("DataBindingController.bindData()");
            System.out.println("SNO   is :: "+sno);
            System.out.println("SNAME is :: "+sname);
            return "show_data";
}
output:: Exception

case2:: http://localhost:9999/DataBindingApp/data?sno=10
@GetMapping("/data")
public String bindData(Map<String, Object> model, @RequestParam Integer
sno,@RequestParam(required = false)String sname) {
            System.out.println("DataBindingController.bindData()");
            System.out.println("SNO   is :: "+sno);
            System.out.println("SNAME is :: "+sname);
            return "show_data";
}
output:: sno = 10 sname  =

case3::http://localhost:9999/DataBindingApp/data?sno=10
@GetMapping("/data")
public String bindData(Map<String, Object> model, @RequestParam Integer
sno,@RequestParam(defaultValue="sachin") String sname) {
            System.out.println("DataBindingController.bindData()");
            System.out.println("SNO   is :: "+sno);
            System.out.println("SNAME is :: "+sname);
            return "show_data";
}
output : sno = 10 sname=sachin
```