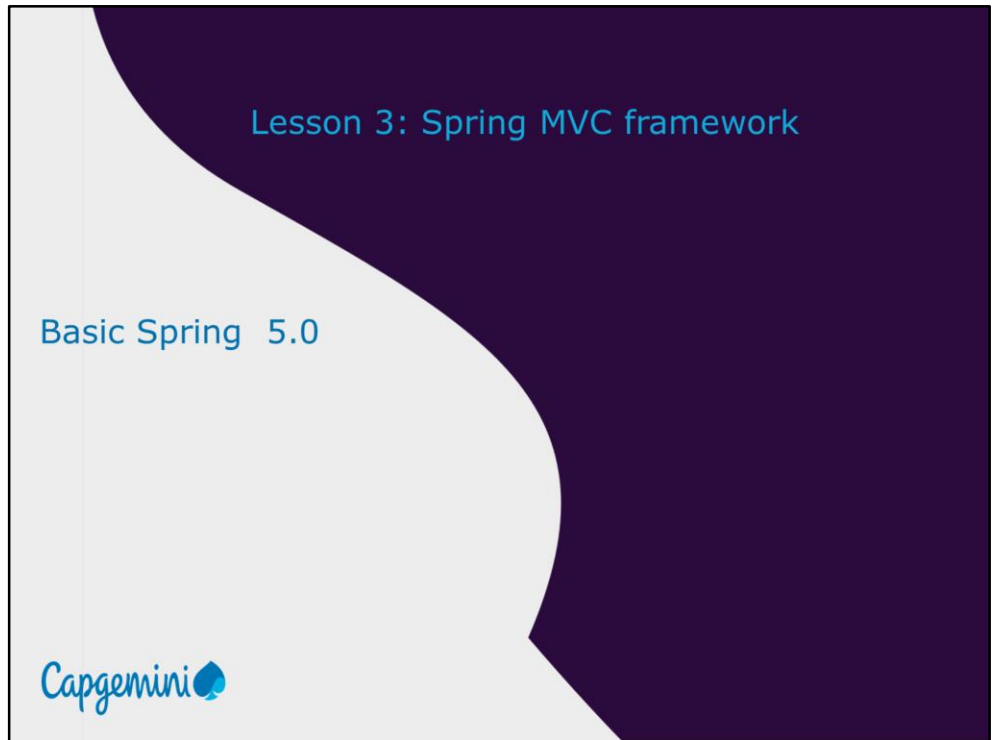**Instructor Notes:**

Add instructor notes here.

Instructor Notes:

Lesson Objectives

Capgemini

Web applications have become a very important part of any enterprise system. The key requirements for a web framework is to simplify development of the web tier as much as possible. Spring provides a web framework based on the MVC (Model view Controller) paradigm. Although it is similar in some ways to other popular MVC frameworks such as Struts and WebWork, Spring web MVC provides significant advantages over those frameworks.

Spring MVC helps in building flexible and loosely coupled web applications. The Model-view-controller design pattern helps in separating the business logic, presentation logic and navigation logic. Models are responsible for encapsulating the application data. The Views render response to the user with the help of the model object . Controllers are responsible for receiving the request from the user and calling the back-end services.

Instructor Notes:

3.1:Spring MVC introduction

- MVC design pattern
- Dispatcher Servlet – Front Controller
- Controllers
- Request Handler
- ModelAndView
- ViewResolver

Capgemini

The Spring MVC is a web framework built within the Spring Framework. There are a number of challenges while creating a web-based application - like state management, workflow and validation, which are addressed by Spring's web framework. This framework can be used to automatically populate your model objects from incoming request parameters while providing validation and error handling as well. The entire framework is modular, with each set of components having specific roles and completely decoupled from the rest of the framework. This allows you to develop the front end of your web application in a very pluggable manner.

MVC provides out-of-the-box implementations of workflow typical to web applications. Its highly flexible, allowing you to use a variety of different view technologies. It also enables you to fully integrate with your Spring based, middle-tier logic through the use of dependency injection.
You can use Spring web MVC to make services that are created with other parts of Spring available to your users, by implementing web interfaces.

Instructor Notes:

3.1 : Spring MVC Architecture
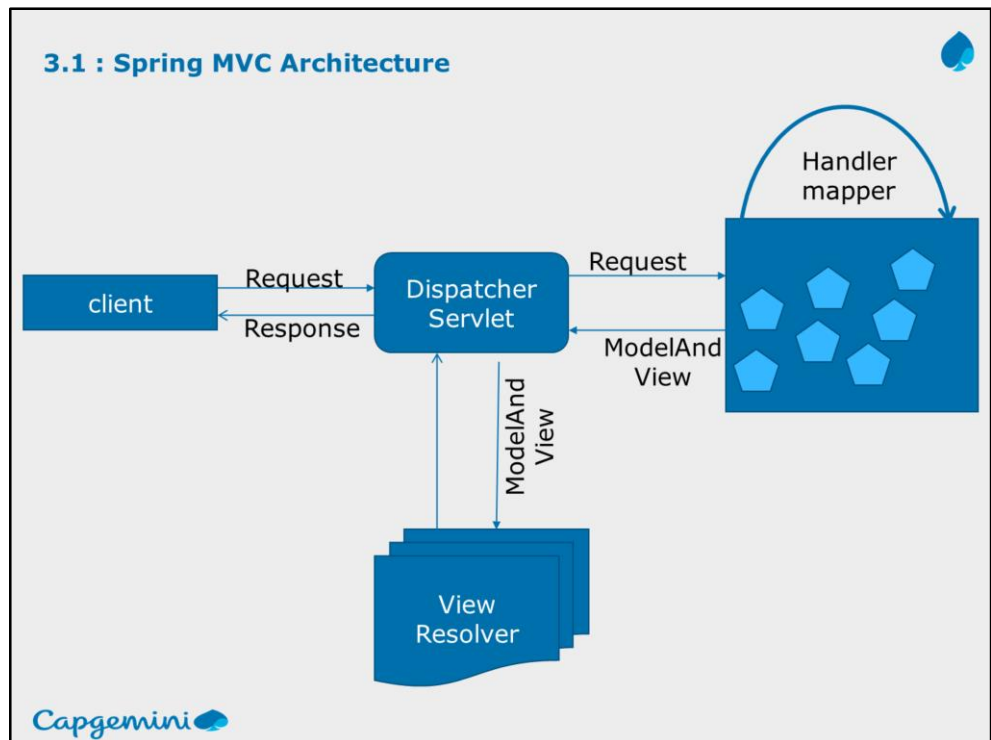
Life cycle of a request in Spring MVC: From the time that a request is received by Spring until the time that a response is returned to the client, many pieces of Spring MVC framework are involved.

The process starts when a client (typically a web browser) sends a request. It is first received by a DispatcherServlet. Like most Java-based MVC frameworks, Spring MVC uses a front-controller servlet (here DispatcherServlet) to intercept requests. This in turn delegates responsibility for a request to other components of an application for actual processing.

The Spring MVC uses a Controller component for handling the request. But a typical application may have several controllers. To determine which controller should handle the request, DispatcherServlet starts by querying one or more HandlerMappings. A HandlerMapping typically maps URL patterns to Controller objects.

Once the DispatcherServlet has a Controller object, it dispatches the request to the Controller which performs the business logic (a well-designed Controller object delegates responsibility of business logic to one or more service objects). Upon completion of business logic, the Controller returns a ModelAndView object to the DispatcherServlet. The ModelAndView object contains both the model data and logical name of view. The DispatcherServlet queries a ViewResolver with this logical name to help find the actual JSP. Finally the DispatcherServlet dispatches the request to the View object, which is responsible for rendering a response back to the client.

Instructor Notes:

3.1 : Configuring DispatcherServlet in web.xml

```
<servlet>
   <servlet-name>basicspring</servlet-name>
   <servlet-class> org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
</servlet>


<servlet-mapping>
   <servlet-name>basicspring</servlet-name>
   <url-pattern>*.obj</url-pattern>
</servlet-mapping>
```

The servlet-name given to the servlet is significant

Capgemini

Configuring the DispatcherServlet
The dispatcher servlet is at the heart of the Spring MVC and functions as Spring MVC's front controller. Like any other servlet, it must be configured in web.xml file. Place the <servlet> declaration (in the first listing above) in the web.xml file. The servlet-name given to the servlet is significant. By default, when DispatcherServlet is loaded, it will load the Spring application context from an xml file whose name is based on the name of the servlet. In the above example, the servlet-name is basicspring and so the DispatcherServlet will try to load the application context from a file called basicspring-servlet.xml.
Next, you must indicate which URL's will be handled by DispatcherServlet. Add the following <servlet-mapping> tag (the second xml listing) to web.xml to let DispatcherServlet handle all url's that end in .obj. The URL pattern is arbitrary and could be any extension.
DispatcherServlet is now configured and ready to dispatch requests to the web layer of your application.

Pls. see the above steps that are followed while building a simpleMVC application. These steps are explained in detail in the coming slides.

Instructor Notes:

3.1 : WebApplicationContext

```xml
<listener>
  <listener-class>
            org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

```xml
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
      /WEB-INF/basicspring-service.xml
      /WEB-INF/basicspring-data.xml
   </param-value>
</context-param>
```

Capgemini

We have seen that DispatcherServlet will load the Spring application context from a single XML file whose name is <servlet-name>-servlet.xml. But you can also split the application context across multiple XML files. Ideally splitting it into logical pieces across application layers can make maintenance easier by keeping each of the Spring configuration files focused on a single layer of the application. To ensure that all the configuration files are loaded, you will need to configure a context loader in your web.xml file. A context loader loads context configuration files in addition to the one that DispatcherServlet loads. The most commonly used context loader is a servlet listener called ContextLoaderListener that is configured in web.xml as shown in the first listing above.

With ContextLoaderListener configured, we need to tell it the location of the Spring configuration file(s) to load. If not specified, context loader will look for a Spring configuration file at /WEB-INF/applicationContext.xml. We will specify one or more Spring configuration file(s) by setting the contextConfigLocation parameter in the servlet context as seen in the second listing above. The context loader will use contextConfigLocation to load the two context configuration files – one each for the service and data layer.

Instructor Notes:

3.2 Annotation-based configuration – Controller

```
@Controller
 public class HelloController {
    @RequestMapping("/helloWorld")
    public String showMessage() {
                return "hello";
}}
```

Capgemini

Instructor Notes:

3.2:Annotation-based controller configuration

- @Controller

- @RequestMapping

- @RequestParam

- @ModelAttribute

Capgemini

@Controller: Indicates that an annotated class is a "Controller"

Example:
@Controller
Public class LoginController{}

@RequestMapping: @RequestMapping can be applied to the controller class
as well as methods. It maps web requests onto specific handler classes and/or
handler methods.

Example:
@Controller
@RequestMapping("loginController")
Public class LoginController{
            @RequestMapping("loadForm")
            public String loadData(){}
}

URL need to be used for making the request for the above mentioned controller
is

http://localhost:8080/projectname/loginController/loadForm.obj

Instructor Notes:

3.2.1 Handler Mapping

- Handler mapping bean in the **WebApplicationContext** that implements the HandlerMapping interface.

- Map a request to a handler according to the request's URL.

- Use **@RequestMapping** annotation to identify the services in controller.

```
@Controller
public class MyController{

        @RequestMapping("/")
        public ModelAndView sayHello(){
                return new ModelAndView("hello",'msg',"Hello World");
        }

}
```

Capgemini

Instructor Notes:

3.2.1 ModelAndView

```
new ModelAndView("viewName","modelObjectName","modelObject");
```

```
Map myModel = new HashMap();
myModel.put("now",now);
myModel.put("products",getProductManager().getProducts());
return new ModelAndView("product","model",myModel);
```

Capgemini

The ModelAndView class allows you to specify a response to the client, resulting from actions performed in controller. This object holds both – the view the client will be presented with and the model used to render the view. Every controller must return a ModelAndView. See the two examples above.
The first parameter is the logical name of a view component that will be used to display the output from this controller. The next two parameters represent the model object that will be passed to the view and its value.
In the second example, the view component is product and the model is an object that is to be returned and its value is MyModel which is a HashMap object containing multiple values.
In the end, the dispatcher servlet needs a concrete view instance to render the view.

Instructor Notes:

3.2.1 Building a basic Spring MVC application – ViewResolver

- DispatcherServlet receives a model and a view name, it will resolve the logical view name into a view object for rendering.

- DispatcherServlet resolves views from one or more view resolvers.

- A view resolver is a bean configured in the **WebApplicationContext** that implements the **ViewResolver** interface.

- Its responsibility is to return a view object for a logical view name.

Capgemini

Instructor Notes:

3.2.1 Building a basic Spring MVC application -
Resolving Views: The ViewResolver

- **InternalResourceViewresolver**:

    Resolves logical view names into View objects that are rendered using

    template file resources

- **BeanNameViewResolver**:

    Looks up implementations of the View interface as beans in the Spring

    context, assuming that the bean name is the logical view name

- **ResourceBundleViewResolver**

    Uses a resource bundle that maps logical view names to

    implementations of the View interface

- **XmlViewResolver**

    Resolves View beans from an XML file that is defined separately from

    the application context definition files

Capgemini

So far, we have seen how model objects are passed to the view through the
ModelAndView object. In Spring MVC, a view is a bean that renders results to
the user. The view most likely is a JSP. But you could also use other view
technologies like Velocity and FreeMarker templates or even views that
produce PDF and MS-Excel documents.

View resolvers resolve the view name given by the ModelAndView object to a
View bean. Spring provides a number of useful view resolvers, some of which
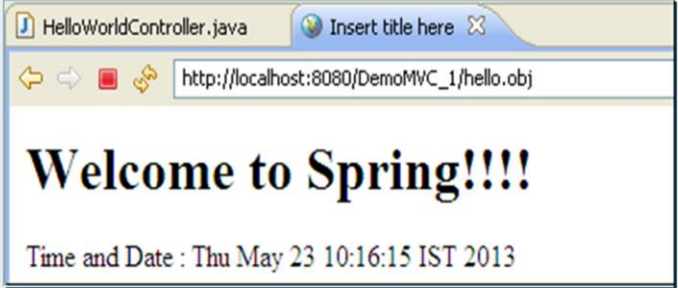are shown in the table above. See Spring docs for more.

InternalResourceViewResolver resolves a logical view name by affixing a prefix
and a suffix to the view name returned by the ModelAndView object. It then
loads a View object with the path of the resultant JSP. By default, the view
object is an InternalResourceView, which simply dispatches the request to the
JSP to perform the actual rendering. But, if the JSP uses JSTL tags, then you
may replace InternalResourceView with JstlView as seen in the code demos
earlier.

```xml
<bean id="viewResolver" class=
"org.springframework.web.servlet.view.InternalResourceViewResolver ">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView </value>
  </property>
  <property name="prefix"><value>/</value></property>
  <property name="suffix"><value>.jsp</value></property>
</bean>
```

Instructor Notes:

Demo

- Refer DemoMVC_1

HelloWorldController.java	Insert title here

http://localhost:8080/DemoMVC_1/hello.obj

Welcome to Spring!!!!

Time and Date : Thu May 23 10:16:15 IST 2013

Capgemini

Please refer to the DemoMVC_1 web project.

Instructor Notes:

Demo 7->2

Demo

- Refer DemoMVC_2 application

Capgemini

Please refer to the DemoMVC_2web project.
Refer to HelloWorldController.java.
Invoke the application as :
http://localhost:8080/DemoMVC_2/hi/hello.obj?name=CapGemini

Instructor Notes:

3.2.2: Spring MVC annotations -Validating input with Bean Validation

- Bean Validation (JSR – 303) Annotations:

Annotation Name	Description
Annotations for validation	
@Valid	To trigger validation of a @Controller input
@Size	Validates that the fields meet criteria on their length.
@NotNull	Validates that the fields contains value.
@Pattern	@Pattern annotation along with a regular expression ensures that the entered value is valid
@Email	Validates that the field value is a valid emailid.
@DateTimeFormat	In Spring New Date & Time API can be used in Controllers for Form Binding

Capgemini

Before an object can be processed further, it is essential to ensure that all the data in the object is valid and complete. Faulty form input must be rejected. For example, username must not contain spaces, password must be minimum 6 characters long, email must be correct etc.

The @Valid annotation (part of the JavaBean validation specification) tells Spring that the User object should be validated as it's bound to the form input. If anything goes wrong while validating the User object, the validation error will be carried to the processForm() method via the BindingResult that's passed in on the second parameter. If the BindingResult's hasErrors() method returns true, then that means that validation failed.

How do we declare validation rules?

JSR-303 defines some annotations that can be placed on properties to specify validation rules. The code above shows the properties of the User class that are annotated with validation annotations.

@Size annotation validates that the fields meet criteria on their length.

@Pattern annotation along with a regular expression ensures that the value given to the email property fits the format of an email address and that the username is only made up of alphanumeric characters with no spaces.

Notice how we've set the message attribute with the message to be displayed in the form when validation fails. With these annotations, when a user submits a registration form to AddUserFormController's processForm() method, the values in the User object's fields will be validated. If any of those rules are violated, then the handler method will send the user back to the form.

Instructor Notes:

3.2.2: Spring MVC annotations -Validating input : declaring validation rules

```
public class User {
    @Size(min = 3, max = 20, message = "Username must be between 3 and 20
characters long.")
    @Pattern(regexp = "^[a-zA-Z0-9]+$", message = "Username must be
alphanumeric with no spaces")
    private String username;

    @Size(min = 6, max = 20, message = "The password must be at least 6
characters long.")
    private String password;

    @Pattern(regexp = "[A-Za-z0-9]+@[A-Za-z0-9.-]+[.][A-Za-z]{2,4}",
message = "Invalid email address.")
    private String email;

//getter and setter methods for all these properties
}
```

Capgemini

Instructor Notes:

3.2.2 : Spring MVC annotations -Processing forms : The JSP

addUser.jsp

```jsp
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
<sf:form method="POST" modelAttribute="user" >
<table cellspacing="0">
<tr>
<th><sf:label path="username">Username:</sf:label></th>
<td><sf:input path="username" size="15" maxlength="15" />
     <small id="username_msg">No spaces, please.</small><br />
     <sf:errors path="username" /></td>
</tr>
<tr>
    <th><sf:label path="password">Password:</sf:label></th>
    <td><sf:password path="password" size="30" showPassword="true"/>
      <small>6 characters or more (be tricky!)</small><br/>
      <sf:errors path="password" />
      </td>
   </tr>
<tr><th></th>
<td><input name="commit" type="submit" value="Save User" /></td></tr>
</sf:form></div>
```

Capgemini

The addUser.jsp page uses Spring's form binding library. The <sf:form> tag binds the User object (identified by the modelAttribute attribute - see above) that showForm() placed into the model to the various fields in the form. The <sf:input> and <sf:password> tags have a path attribute that references the property of the User object that the form is bound to. When the form is submitted, whatever values these fields contain will be placed into a User object and submitted to the server for processing.

Note that the <sf:form> specifies that it'll be submitted as an HTTP POST request. We thus now need another handler method that accepts POST requests. See the processForm() method in the code listing in the previous page which will process form submissions.

When the addUser.jsp form is submitted, the fields in the request will be bound to the User object (passed as an argument to processForm(). From there, some logic can be employed to persist user object into database. Notice that User parameter is annotated with @Valid. This indicates that the User should pass validation before being persisted. We shall cover validation next.

Instructor Notes:

3.2.2 : Spring MVC annotations

- Displaying validation errors

```
<td>                                                          jsp
    <sf:password path="password" size="30"
showPassword="true"/>
          <small>6 characters or more (be
tricky!)</small><br/>
          <sf:errors path="password" />
</td>
```

```
                                                          controller
public String processForm(@Valid User user, BindingResult
bindingResult) {
    if (bindingResult.hasErrors()) {
        return "failure";
    }
.....
```

Capgemini

We now need to tell jsp to display the validation messages.
We have passed the BindingResult parameter to processForm(). This knows
whether the form had any validation errors via its hasErrors() method. But the
actual error messages are also in there, associated with the fields that failed
validation. To display those errors to the users, use Spring's form binding JSP
tag library to display the errors. ie, the <sf:errors> tag can render field
validation errors. See code listing above to see how this is done.
The <sf:errors> tag's path attribute specifies the form field for which errors
should be displayed. For example, the above listing displays errors (if any) for
the field whose name is password. If there are multiple errors for a single field,
they'll all be displayed, separated by an HTML <br/> tag.

Instructor Notes:

3.2.2 : Spring MVC annotations - Processing forms :

The controller class

```
@Controller
public class AddUserFormController {
  @RequestMapping(value = "/AddUser", method = RequestMethod.GET)
    public String showForm(Model model) {
        model.addAttribute(new User());
        return "addUser";
    }
  @RequestMapping(method = RequestMethod.POST)
    public String processForm(@Valid User user, BindingResult bindingResult) {
        if (bindingResult.hasErrors())   return "failure";
        else {
            // some logic to persist user
            return "success";
}}
```

Create a User

Username:		No spaces, please.
Password:		6 characters or more (be tricky!)
Email Address:		In case you forget something

Save User

addUser.jsp

Capgemini

Working with forms in a web application involves two operations: displaying the form and processing the form submission.
Our example allows to register new User. For this we have defined two handler methods to AddUserFormController to handle each of the operations.
We first need to display the form in the browser before it can be submitted. The first method (showForm()) displays the registration form. See the figure above.
Once the form is displayed, it'll need a User object to bind to the form fields. The showForm() method creates a User object and places it in the model.

Instructor Notes:

3.2.3 Dispatcher Servlet Java Based Configuration

DispatcherServlet can be configured programmatically by implementing or extending either of these three support classes provided by Spring –

- WebAppInitializer interface
- AbstractDispatcherServletInitializer abstract class
- AbstractAnnotationConfigDispatcherServletInitializer abstract class

- WebApplicationInitializer is a perfect fit for use with Spring's code-based @Configuration classes.

Capgemini

Instructor Notes:

3.2.3 Dispatcher Servlet Java Based Configuration

```java
public class MyWebAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext container) {
     // Create the 'root' Spring application context
     AnnotationConfigWebApplicationContext rootContext =
       new AnnotationConfigWebApplicationContext();
     rootContext.register(AppConfig.class);

     // Manage the lifecycle of the root application context
     container.addListener(new ContextLoaderListener(rootContext));
   }
}
```

Capgemini

Instructor Notes:

3.2.3 Dispatcher Servlet Java Based Configuration

```
// Create the dispatcher servlet's Spring application context
    AnnotationConfigWebApplicationContext dispatcherContext =
      new AnnotationConfigWebApplicationContext();
    dispatcherContext.register(DispatcherConfig.class);


    // Register and map the dispatcher servlet
    ServletRegistration.Dynamic dispatcher =
      container.addServlet("dispatcher", new    DispatcherServlet(dispatcherContext));
    dispatcher.setLoadOnStartup(1);
    dispatcher.addMapping("/");
  }
```

Capgemini

Instructor Notes:

3.2.4 Spring 5 MVC Annotations

@Controller :

    It will make class as a request handler.

@GetMapping :

    It is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). @GetMapping annotated methods handle the HTTP GET requests matched with given URI expression

@PostMapping :

    It is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST). @PostMapping annotated methods handle the HTTP POST requests matched with given URI expression.

@EnableWebMvc

Other annotations like **@PutMapping, @DeleteMapping, @PatchMapping** can be used in similar way.

Capgemini

```
package com.cg.controller;

@Controller
public class HomeController
{
   @GetMapping("/")
   public String homeInit(Model model) {
      return "home";
   }
   @GetMapping("/home")
public String homeInit(Model model) {
   return "home";
}

@GetMapping("/members/{id}")
public String getMembers(Model model) {
   return "member";
}
@PostMapping(path = "/members", consumes = "application/json", produces
="application/json")
public void addMember(@RequestBody Member member) {
   //code
}

}
```

Instructor Notes:

Demo 7->2

Demo

- Refer MVCJavaBasedExample application

Capgemini

Instructor Notes:

Additional notes for
instructor

Demo

- Refer the following Demos:
    - DemoMVC_3
    - DemoMVC_4
    - DemoMVC_5
    - DemoMVC_6
    - DemoMVC_Complete
    - MVCJavaBasedExample

Capgemini

Instructor Notes:

Additional notes for instructor

Summary

- How to use Spring MVC architecture to build flexible and powerful web applications.
- Components like handler mappings, ViewResolvers and controllers
- MVC Annotations like @Controller, @RestController, @RequestMapping , @RequestParam, @PathVariable
- Spring 5 MVC Annotations like @GetMapping, @PostMapping,@EnableWebMvc
- Spring 5 MVC Java based Web Application

Summary

Capgemini

Instructor Notes:

Ans-1 : 1. Pls. explain, if necessary, to participants using the older technique's config file. Example is available therein.

Ans-2 : 1

Review Questions

- Question 1: If multiple handler mappings have   been declared in an application, select the property that indicates which handler mapping has precedence?
  - Option 1: Order
  - Option 2: Sequence
  - Option 3: Index
  - Option 4: An application cant have multiple handler mappings
- Question 2: To figure out which controller should handle the request, DispatcherServlet queries _____
  - Option 1: HandlerMappings
  - Option 2: ModelAndView
  - Option 3: ViewResolver
  - Option 4: HomeController

Capgemini