



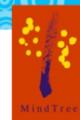
# Spring MVC

Campus Batch 2011

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

## Objectives



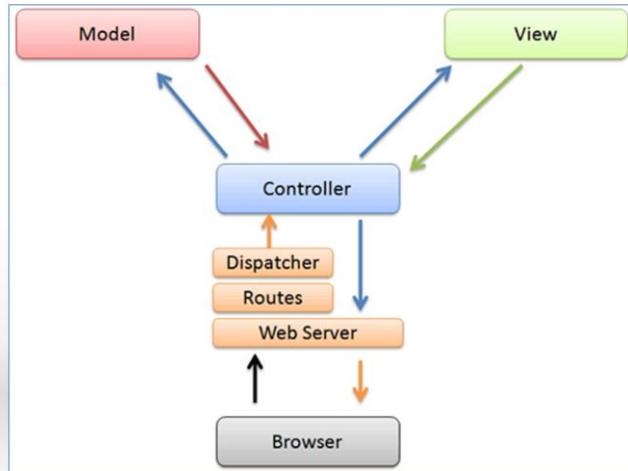
- Overview of MVC Paradigm
- Understand the components of Spring MVC
- Implementing a basic controller
- Creating a Simple view
- Configuring a Spring MVC application
- Understand Spring 3 MVC
- Annotating Controller's and RequestMapping
- Spring MVC Hibernate Integration
- Model driven attribute using @ModelAttribute
- Using Spring Validation for validating form fields
- Understand PropertyEditors for converting string type to custom data type.
- Understand how to pre populate form fields

## Model View Controller (MVC)



- MVC = Model-View-Controller
  - Clearly separates business, navigation and presentation logic
  - Proven mechanism for building a thin and clean web-tier.
- Three core collaborating components
  - Controller
    - Handles navigation logic and interacts with the service tier for business logic
  - Model
    - The contract between the Controller and the View
    - Contains the data needed to render the View
    - Populated by the Controller
  - View
    - Renders the response to the request
    - Pulls data from the model

- MVC Components



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 4

The **browser** makes a request, such as  
<https://konnect.mindtree.com/communities/j2ee/Pages/>

The **web server** (Apache, WebSphere, etc.) receives the request. It uses **routes** to find out which controller to use.

The web server then uses the **dispatcher** to create a new controller, call the action and pass the parameters.

**Controllers** do the work of parsing user requests, data submissions, cookies, sessions and the “browser stuff”. They act as locus for transferring data between Model and View.

**Models** are Java classes. They talk to the database, store and validate data, perform the business logic and otherwise do the heavy lifting.

**Views** are what the user sees: HTML, CSS, XML, Javascript, JSON. **Views are merely puppets** reading what the controller gives them. They don't know what happens in the back room.



### ● Motivation

- Eases maintenance burden
  - Changes to business logic are less likely to break the presentation logic
  - Changes to presentation logic also does not break business logic.
- Facilitates multi-disciplined team development
  - Developers can focus on creating robust business code without having to worry about breaking the UI
  - Designers can focus on building usable and engaging UIs without worrying about Java
- Use the best tool for the job
  - Java is especially suited to creating business logic code
  - Markup or template languages are more suited to creating HTML layouts.
- Ease testability
  - Business and navigation logic are separated from presentation logic meaning they can be tested separately
  - Practically: you can test more code outside the Servlet container



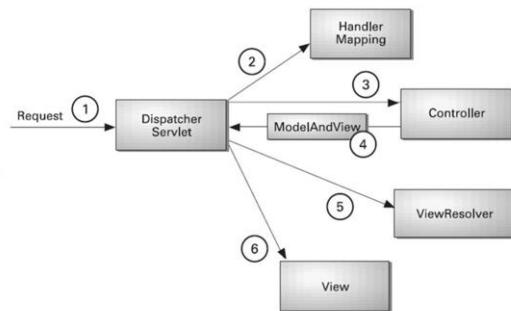
### ● Core Components of Spring MVC

- DispatcherServlet
  - Spring's Front Controller implementation. Request routing is completely controlled by the Front Controller. As an application developer, you will have to just configure the DispatcherServlet in web.xml

- Controller
  - An application developer created component for handling requests.
  - Controllers are POJOs which are managed by Spring ApplicationContext just like any other bean
  - Controllers encapsulates navigation logic.

- View
  - An application developer created pages responsible for rendering output.

## Spring MVC



1. The **DispatcherServlet** first receives the request
2. The **DispatcherServlet** consults the **HandlerMapping** and invokes the **Controller** associated with the request
3. The **Controller** process the request by calling the appropriate service methods
4. The **Controller** returns a **ModeAndView** object to the **DispatcherServlet**. The **ModeAndView** object contains the model data and the view name.
5. The **DispatcherServlet** sends the view name to a **ViewResolver** to find the actual View to invoke.
6. Now the **DispatcherServlet** will pass the model object to the **View** to render the result. The **View** with the help of the model data will render the result back to the user

## Writing your first Spring MVC Web application



### ● Step 1.

- Create a Dynamic Web Project
- Copy Spring jar files to WEB-INF\lib folder

The screenshot shows two windows side-by-side. On the left is the 'New Dynamic Web Project' dialog box. It has a title bar 'New Dynamic Web Project'. Inside, there's a 'Dynamic Web Project' section with instructions: 'Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.' Below this are fields for 'Project name' (set to 'HelloWorldSpringMVC'), 'Project location' (set to 'D:\Testing\HelloWorldSpringMVC'), 'Target runtime' (set to 'Apache Tomcat v6.0'), and 'Dynamic web module version' (set to '2.5'). At the bottom are buttons for '?', '< Back', 'Next >', 'Finish' (which is highlighted with a yellow box), and 'Cancel'. On the right is a file explorer showing the project structure for 'HelloWorldSpringMVC'. The 'WEB-INF\lib' directory is expanded, displaying a list of Spring JAR files:

- org.springframework.aop-3.1.0.M2.jar
- org.springframework.asm-3.1.0.M2.jar
- org.springframework.aspects-3.1.0.M2.jar
- org.springframework.beans-3.1.0.M2.jar
- org.springframework.context-support-3.1.0.M2.jar
- org.springframework.context-3.1.0.M2.jar
- org.springframework.core-3.1.0.M2.jar
- org.springframework.expression-3.1.0.M2.jar
- org.springframework.instrument.tomcat-3.1.0.M2.jar
- org.springframework.instrument-3.1.0.M2.jar
- org.springframework.jdbc-3.1.0.M2.jar
- org.springframework.orm-3.1.0.M2.jar
- org.springframework.test-3.1.0.M2.jar
- org.springframework.transaction-3.1.0.M2.jar
- org.springframework.web.servlet-3.1.0.M2.jar
- org.springframework.web-3.1.0.M2.jar

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 8



### ● Step 2

#### ● Configure DispatcherServlet (FrontController).

- Requests that you want the DispatcherServlet to handle will have to be mapped using a URL mapping in the same web.xml file.
- We have configured all requests ending with “.view” will be handled by the ‘dispatcher’ DispatcherServlet.

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

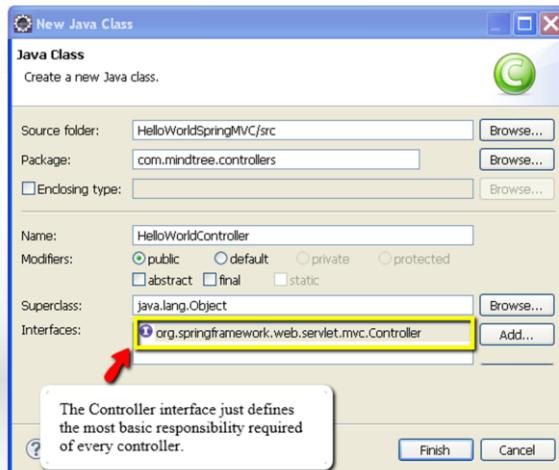
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.view</url-pattern>
</servlet-mapping>
```

## Writing your first Spring MVC Web application



### ● Step 3

#### ● Adding Controller



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 10

Spring's basis for the controller architecture is the `org.springframework.web.servlet.mvc.Controller` interface:

```
public interface Controller {
```

```
    /** * Process the request and return a ModelAndView object which the  
    DispatcherServlet
```

```
        * will render.
```

```
    */
```

```
    ModelAndView handleRequest( HttpServletRequest request,  
    HttpServletResponse response) throws Exception;
```

```
}
```



### ● Step 4.

#### ● Coding your controller

```
/*
 * @author Banu Prakash
 * © 2011 MindTree Limited
 */
public class HelloWorldController implements Controller {
    protected final Logger logger = Logger.getLogger(getClass());
    @Override
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        logger.info("returning hello view with Model data");
        Map<String, Object> model = new HashMap<String, Object>();
        // populate some book list. generally this data comes from service layer
        List<String> books = new ArrayList<String>();
        books.add("Spring in Action");
        books.add("Hibernate in Action");
        books.add("Head First Java");
        // Add date and book information to model
        model.put("now", new Date());
        model.put("bookList", books);
        //return ModelAndView(viewName, modelParameterName, modelParameterValue)
        return new ModelAndView("hello", "model", model);
    }
}
```

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 11

The HelloWorldController stores model data in a Map as “now” and “bookList” keys and returns ModelAndView object which contains “hello” as the name of view and “model” as model parameter name. The ViewResolver uses the view name to forward to appropriate view.



### ● Step 5

#### ● Configure the Controller class

- Here the Servlet name is *dispatcher*. By default the *DispatcherServlet* will look for a file name *dispatcher-servlet.xml* to load the Spring MVC configuration. This file name is formed by concatenating the Servlet name ("dispatcher") with "-servlet.xml".

WEB-INF/web.xml

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

WEB-INF/dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean name="/HelloWorld.view" class="com.mindtree.controllers.HelloWorldController"/>
</beans>
```

BeanNameUrlHandlerMapping maps the bean name "/HelloWorld.view" to HelloWorldController

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 12

By default the *DispatcherServlet* uses the *BeanNameUrlHandlerMapping* to map the incoming request.

The *BeanNameUrlHandlerMapping* uses the bean name as the URL pattern. Since *BeanNameUrlHandlerMapping* is used by default, you need not do any separate configuration for this.

Also in our web.xml we have configured that any url ending with ".view" will be handled by the *DispatcherServlet*.

## Writing your first Spring MVC Web application



### ● Step 6

#### ● Configure ViewResolver

WEB-INF/dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <!-- configure controller -->
    <bean name="/HelloWorld.view" class="com.mindtree.controllers.HelloWorldController" />

    <!-- configure view resolver -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

- When the controller returns “hello” as the view name, the viewResolver adds “/WEB-INF/pages” as prefix to “hello” and adds “.jsp” as suffix.
- The view now becomes “[/WEB-INF/pages/hello.jsp](#)”

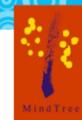
CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 13

Views are represented using logical view names which are returned by the Controller  
Without using ViewResolver, Controllers can return an actual View class if needed.

## Writing your first Spring MVC Web application



### ● Step 7

#### ● Writing view

WEB-INF/pages/hello.jsp

```
<body>
    Date : ${requestScope.model.now}<br />
    Book List : <br />
    <c:forEach items="${requestScope.model.bookList}" var="book">
        <c:out value="${book}" /> <br />
    </c:forEach>
</body>
```

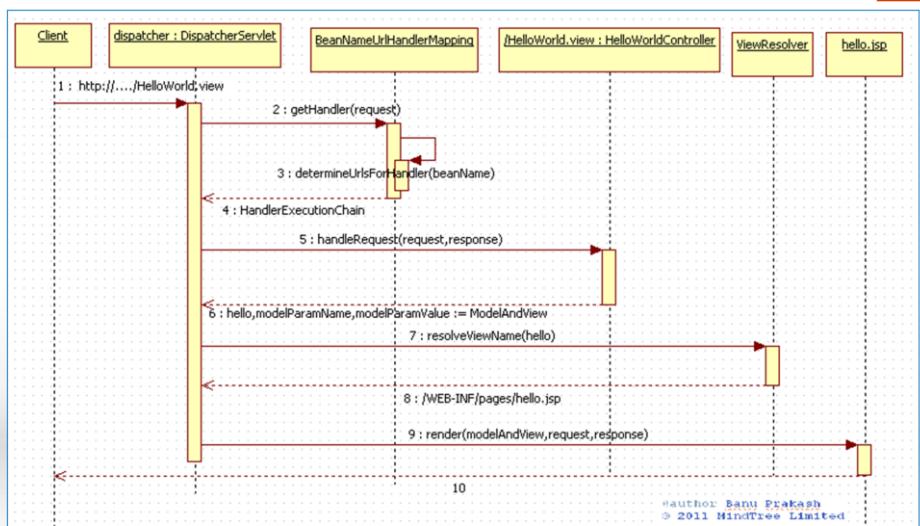
#### ● The HelloWorldController returns

- ModelAndView("hello", "model", model); where "hello" was the view name, "model" was the attribute name stored in request scope.

Browser

Date : Wed Jul 20 15:45:30 IST 2011  
Book List :  
Spring in Action  
Hibernate in Action  
Head First Java

## Sequence diagram for our HelloWorld Spring MVC



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 15

Refer:

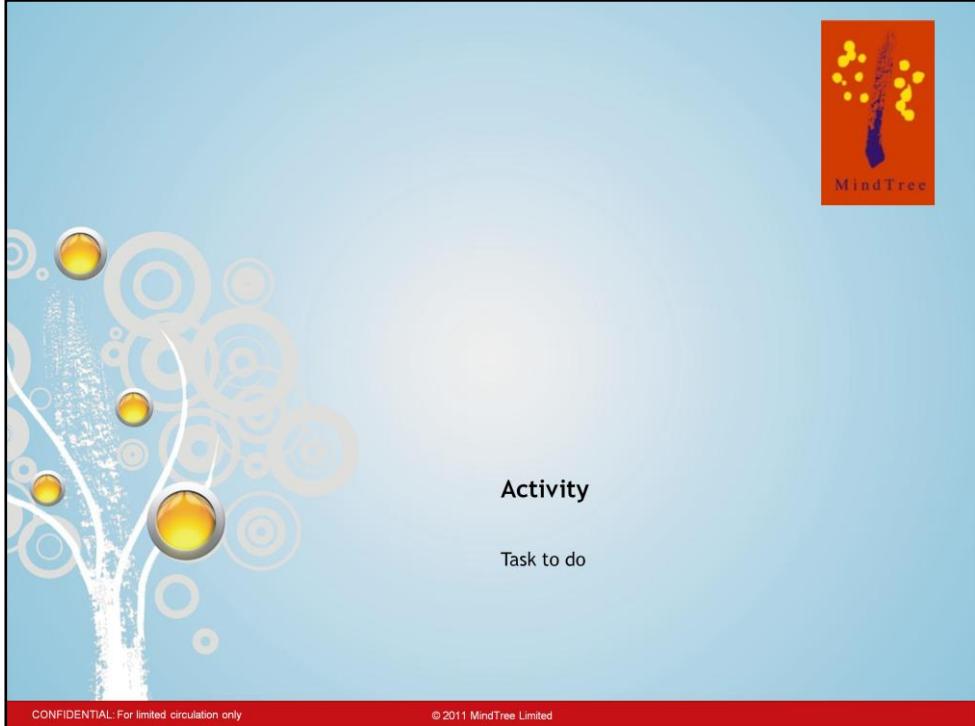
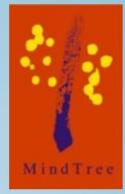
- <http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html>
- <http://static.springsource.org/spring/docs/1.1.5/api/org/springframework/web/servlet/DispatcherServlet.html>
- <http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/web/servlet/HandlerMapping.html>

## Example



### ● Code Example

- **HelloWorldSpringMVC.zip**
  - Example used for the presentation.
  - Illustrates creating a bare minimum Spring MVC application



A decorative slide background featuring a stylized tree on the left side. The tree has white branches and yellow circular leaves. The background is a light blue gradient. In the top right corner, there is a red square containing the MindTree logo. The slide is divided into sections: "Activity" and "Task to do".

Activity

Task to do

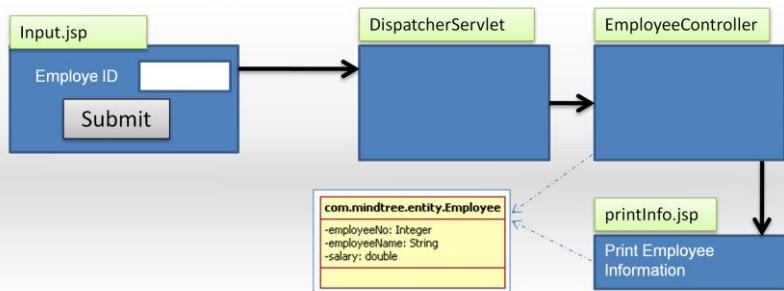
CONFIDENTIAL: For limited circulation only      ©2011 MindTree Limited

Annotation support for Spring started from 2.x, but the Presentation uses 3.x style of Spring MVC



## Activity

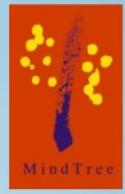
- Spring MVC Activity 1
- Write a Spring MVC application to perform the following:
  - The EmployeeController should have a map having employeeID as key and Employee instance as value
  - User should input employeeID from form field.
  - The EmployeeController should return an ModelAndView containing the Employee instance for the specified employeeID and “printInfo” as the view name.
  - The “printInfo.jsp” should print the employee information.



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 18



## Spring MVC using annotations.

Spring version 3.x style

CONFIDENTIAL: For limited circulation only

©2011 MindTree Limited

Annotation support for Spring started from 2.x, but the Presentation uses 3.x style of Spring MVC



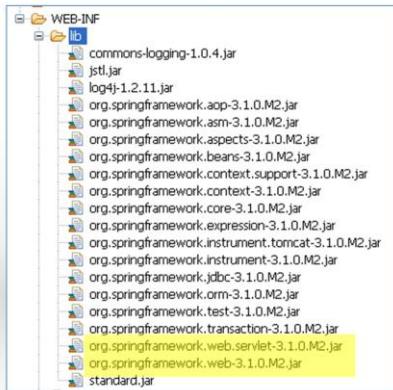
- Why Spring 3 MVC?

- Spring 3 introduces a `mvc` namespace that greatly simplifies Spring MVC setup.
  - Using `mvc` namespace Controllers, ViewResolvers, interceptors and resources configuration becomes that much easier.
- No changes to the `DispatcherServlet` configuration in `web.xml`
- Many other enhancements makes it easier to get Spring 3.x web applications up and running.



### ● Step 1

- Create a Dynamic Web Project
- Add Spring 3 Libraries to WEB-INF/lib folder



Spring libraries  
required for building  
Spring MVC  
application



### ● Step 2

- Configure DispatcherServlet in web.xml [ this remains the same for every version of spring MVC application]

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

- The DispatcherServlet is configured as the default Servlet for the application (mapped to "/")



### ● Step 3

#### ● Configure Controllers and View Resolver

```

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

    <!-- Configures the @Controller programming model -->
    <mvc:annotation-driven />
    <context:component-scan base-package="com.mindtree.controllers"/>

    <!-- Forwards requests to the "/" resource to the "home" view -->
    <mvc:view-controller path="/" view-name="home" />

    <!-- Resolves view names to protected ".jsp" within the /WEB-INF/pages directory -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>

```

Registers the HandlerMapping required to dispatch requests to your @Controllers

<context:component-scan /> tag searches for Controllers which are annotated with @Controller present in the classpath. With annotation based now the Controllers need not implement “Controller” interface and the application developer is free to write any method to handle request, it need not be only handleRequest(request,response).

<mvc:view-controller path="/" view-name="home" /> : Behind the scenes, mvc:view-controller registers a ParameterizableViewController that selects a view for rendering. In this case, when "/" is requested, the "home" view is rendered. The actual view template is a .jsp resolved inside the /WEB-INF/pages directory.



### ● Step 4

- Coding your first controller using annotations

```

package com.mindtree.controllers;

import java.util.Date;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

/**
 * @author Banu Prakash
 * @ 2011 MindTree Limited
 */
@Controller
public class HelloWorldController {
    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("message", "Hello World!");
        mav.addObject("time", new Date());
        return mav;
    }
}

```

**@Controller annotation** allows for auto detection of Controller

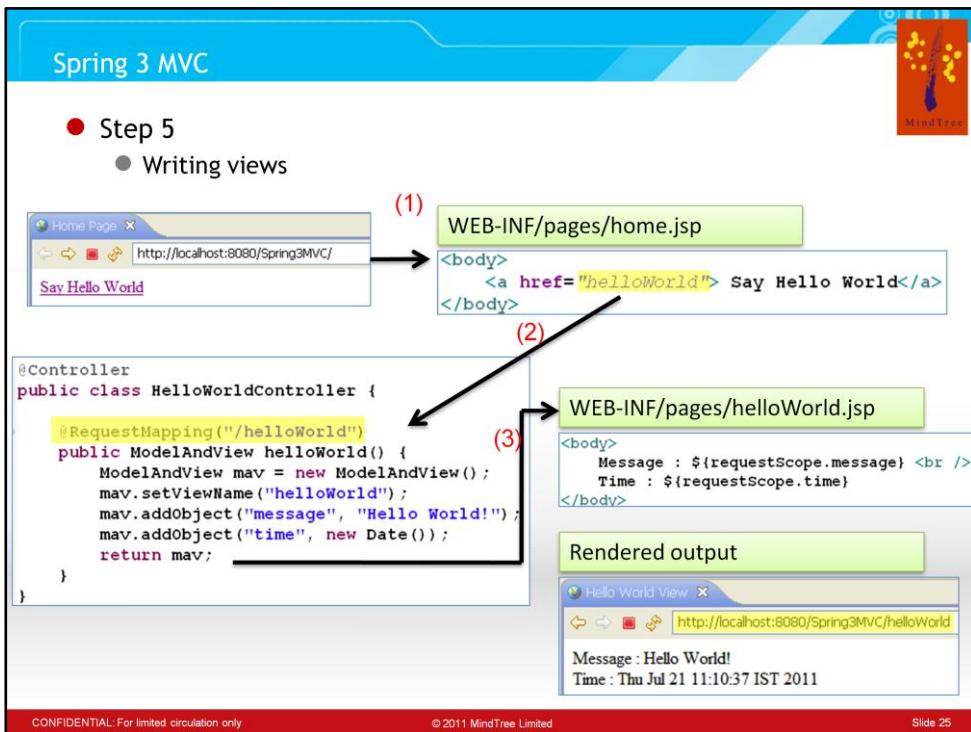
**@RequestMapping("path")** specifies that the method is invoked to handle the request path.

Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces.

Furthermore, they do not usually have direct dependencies on Servlet APIs, although you can easily configure access to Servlet facilities.

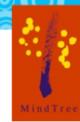
The **@Controller** and **@RequestMapping** annotations allow flexible method names and signatures.

In this particular example the method has no parameters and returns a **ModelAndView**, but various other (and better) strategies exist, as explained later.



1. Client makes a request from browser : “<http://localhost:8080/Spring3MVC/>”, this URL is mapped to `<mvc:view-controller path="/" view-name="home" />` in `dispatcher-servlet.xml`. Using `InternalResourceViewResolver` renders `WEB-INF/pages/home.jsp` page
2. On clicking “Say Hello World” hyperlink, the `helloWorld()`

## Example



- **Code Example**
  - Spring3MVC.zip
    - Example code used for the presentation slides.
- **Reference:**
  - <http://www.roseindia.net/tutorial/spring/spring3/web/spring-3-mvc-hello-world.html>
    - Illustrates step by step creation of Spring MVC application

## Activity

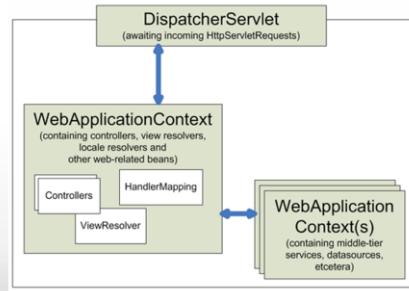


- Spring MVC Activity 2
  - Repeat the Spring MVC Activity 1 using Spring 3 MVC

## Spring MVC - Business Layer integration



- The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications





- **ContextLoaderListener** a servlet listener which is responsible for loading additional WebApplicationContext mostly consisting of beans for service layer and dao layer.

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

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/beans.xml</param-value>
</context-param>
```

beans.xml file consists of all the Dao/Service Layer configurations as discussed in Hibernate Spring integration

- The ContextLoaderListener looks for /WEB-INF/applicationContext.xml by default, but you can override it using the context parameter contextConfigLocation as shown.



- Coding the Controller to interact with service layer

```
/*
 * @author Banu Prakash
 * @ 2011 MindTree Limited
 */
@Controller
public class BankController {

    @Autowired
    private BankService bankService;

    @RequestMapping("/getAccounts")
    public String getAccounts(Model model) {
        String target = "printAccounts";
        try {
            model.addAttribute("accountList",
                bankService.getAllAccounts());
        } catch (ServiceException e) {
            model.addAttribute("errorMessage", e.getMessage());
            target = "home";
        }
        return target;
    }
}
```

## Example



- **Code Example:**

- **Spring3MVC\_Hibernate.zip**

- Illustrates integrating Spring MVC with Hibernate
    - Illustrates configuring ContextLoaderListener to create a WebApplicationContext using “beans.xml” configuration file.
    - Code fetches all accounts present in database using Hibernate API.



- **Request Mapping**

- Method Level mapping
- **By HTTP method**
  - @RequestMapping("path", method=RequestMethod.GET)
  - POST, PUT, DELETE, OPTIONS, and TRACE are also supported
- **By presence of query parameter**
  - @RequestMapping("path", method=RequestMethod.GET, params="foo")
  - Negation also supported: params={ "foo", "!bar" })
- Class Level Mapping is also supported

## Mapping requests



- Configuring MultiAction Controllers

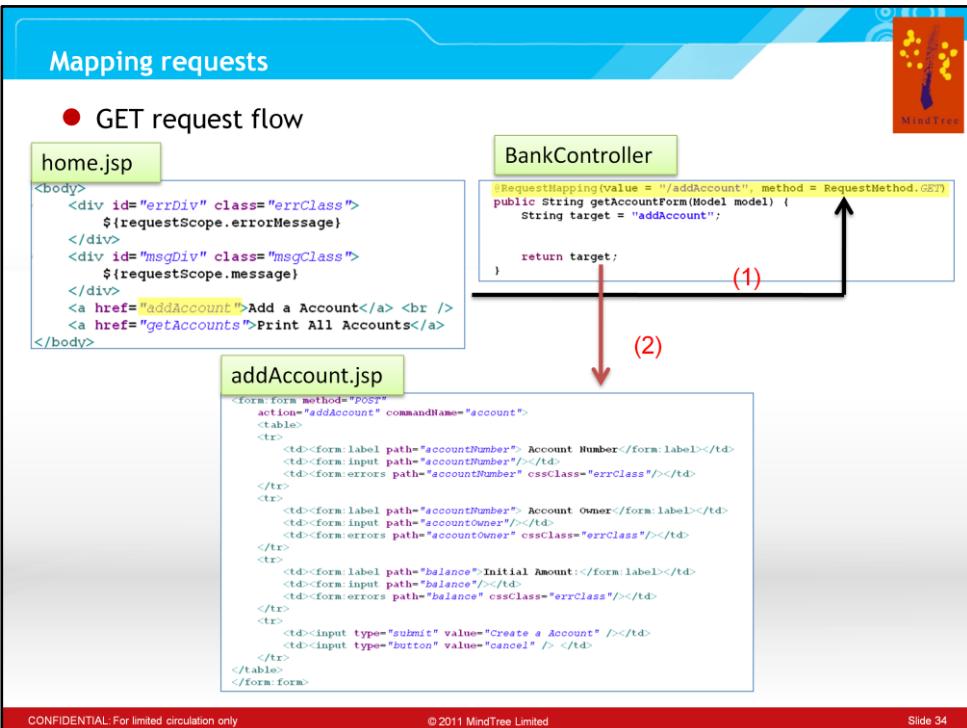
- A Single controller can handle requests for different URL's

```
@RequestMapping(value = "/addAccount", method = RequestMethod.GET)
public String getAccountForm(Model model) {
    String target = "addAccount";
    Remaining code removed for clarity
    return target;
}

@RequestMapping(value = "/addAccount", method = RequestMethod.POST)
public String addAccount(@ModelAttribute("account") Account account,
                        BindingResult errorResults, Model model) {
    String target = "printAccounts";
    Remaining code removed for clarity
    return target;
}
```

For request URI of "/addAccount" and GET request, getAccountForm method is called

For request URI of "/addAccount" and POST request, addAccount method is called



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 34

- 1) Clicking on the hyperlink “Add a Account” GET request is made to the controller and invokes a handler which is configured to handle request mapping “/addAccount” and method=RequestMethod.GET.
- 2) From the `getAccountForm()` method when “addAccount” view name is returned, ViewResolver will render `addAccount.jsp`.



## Mapping requests

### ● POST request flow

#### addAccount.jsp

```
<form:form method="POST"
action="#{addAccount}" commandName="account">
<table>
<tr>
<td><form:label path="accountNumber"> Account Number</form:label></td>
<td><form:input path="accountNumber"/></td>
<td><form:errors path="accountNumber" cssClass="errClass"/></td>
</tr>
<tr>
<td><form:label path="accountOwner"> Account owner</form:label></td>
<td><form:input path="accountOwner"/></td>
<td><form:errors path="accountOwner" cssClass="errClass"/></td>
</tr>
<tr>
<td><form:label path="balance">Initial Amount:</form:label></td>
<td><form:input path="balance"/></td>
<td><form:errors path="balance" cssClass="errClass"/></td>
</tr>
<tr>
<td><input type="submit" value="Create a Account" /></td>
<td><input type="button" value="cancel" /></td>
</tr>
</table>
</form:form>
```

```
@RequestMapping(value = "/addAccount", method = RequestMethod.POST)
public String addAccount(@ModelAttribute("account") Account account,
BindingResult errorResults, Model model) {
    String target = "printAccounts";

    return target;
}
```

When the form is submitted, since the method of request is “POST” addAccount() is called and not getAccountForm().

Remember both addAccount() and getAccountForm() are mapped to same URI.

## @ModelAttribute

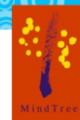


- @ModelAttribute maps a model attribute to the specific, annotated method parameter.
  - This is how the controller gets a reference to the object holding the data auto-populated from request parameters entered in the form.

```
<form:form method="POST"
    action="/addAccount" commandName="account">
    Account Number<form:input path="accountNumber"/> <br/>
    Account Owner<form:input path="accountOwner"/> <br />
    Initial Amount:<form:input path="balance"/> <br />
    <input type="submit" value="Create a Account" />
    <input type="button" value="cancel" />
</form:form>
```

```
@RequestMapping(value = "/addAccount", method = RequestMethod.POST)
public String addAccount(@ModelAttribute("account") Account account, Model model) {
    String target = "printAccounts";
    try {
        bankService.addAccount(account);
    } catch (ServiceException e) {
        model.addAttribute("errorMessage", e.getMessage());
        target = "home";
    }
    return target;
}
```

## Example



- **Code Example:**

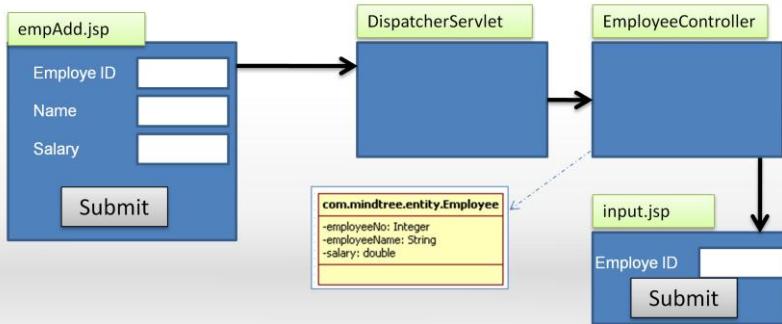
- Refer: Spring3MVC\_Hibernate\_CRUD.zip
  - Illustrates using @RequestMapping annotation to map a URI to a controller method
  - Illustrates using @ModelAttribute annotation to map form field data to entity class

## Activity



### ● Activity 3

- Modify the Activity 2 to add Employee entity to store in a map using @ModelAttribute





- **Validator interface**

- Spring's features a Validator interface that you can use to validate objects.
- The Validator interface works using an Errors object so that while validating, validators can report validation failures to the Errors object.
- Methods of org.springframework.validation.Validator interface:
  - boolean supports(Class)
    - Can this Validator validate instances of the supplied Class?
  - void validate(Object, org.springframework.validation.Errors)
    - validates the given object and in case of validation errors, registers those with the given Errors object

## Validation



### ● Validator Example

```
/**  
 * @author Banu Prakash  
 * @ 2011 MindTree Limited  
 */  
public class AccountValidator implements Validator {  
    /* (non-Javadoc)  
     * @see org.springframework.validation.Validator#supports(java.lang.Class)  
     */  
    @Override  
    public boolean supports(Class<?> clazz) {  
        return clazz.isAssignableFrom(Account.class);  
    }  
  
    /* (non-Javadoc)  
     * @see  
     */  
    public void validate(Object model, Errors errors) {  
        ValidationUtils.rejectIfEmpty(errors, "accountNumber",  
            "acc.No", "Account Number is required");  
        ValidationUtils.rejectIfEmpty(errors, "accountOwner",  
            "acc.Owner", "Account Owner is required");  
        ValidationUtils.rejectIfEmpty(errors, "balance",  
            "acc.Balance", "Account Initial Balance is required");  
        Account account = (Account) model;  
        if(account.getBalance() <= 0) {  
            errors.rejectValue("balance",  
                "acc.balanceInvalid",  
                "Account Initial Balance should be more than zero");  
        }  
    }  
}
```

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 40

ValidationUtils class is used to reject the properties are null or the empty string.

public static void **rejectIfEmpty**(Errors errors, String field, String errorCode, String defaultMessage)

Reject the given field with the given error code and message if the value is empty. The object to validate does not have to be passed in, as the Errors instance allows to check field values (it will usually hold an internal reference to the target object).

**Parameters:** errors Errors instance to register errors on

field - the field name to check

errorCode - error code, interpretable as message key

defaultMessage - fallback default message

Refer:

<http://static.springsource.org/spring/docs/1.1.5/api/org/springframework/validation/ValidationUtils.html>



### ● BindingResult

- Binding and validation errors can be trapped and introspected by declaring a BindingResult parameter
- Must follow the JavaBean parameter in the method signature
- Errors automatically exported in the model when rendering views

```
@RequestMapping(value = "/addAccount", method = RequestMethod.POST)
public String addAccount(@ModelAttribute("account") Account account,
                        BindingResult errorResults, Model model) {
    String target = "printAccounts";
    try {
        validator.validate(account, errorResults);
        if (errorResults.hasErrors()) {
            target = "addAccount";
        } else {
            bankService.addAccount(account);
            model.addAttribute("accountList", bankService.getAllAccounts());
        }
    } catch (ServiceException e) {
        model.addAttribute("errorMessage", e.getMessage());
        target = "home";
    }
    return target;
}
```

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 41

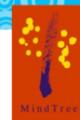
All the validation errors for the model “account” will be stored in errorResults [ BindingResults ].

If errorResults contains any validation errors the user is redirect to the input page “addAccount.jsp”.

In “addAccount.jsp” errors are displayed using <form:errors path=“*propertyName*” cssClass=“*errClass*”/> example <form:errors path=“accountNumber” cssClass=“*errClass*”/>.

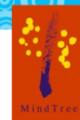
*Note: It is important to note that* the Errors or BindingResult parameters have to follow the model object that is being bound immediately as the method signature.

## Example



- **Code Example:**

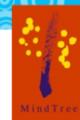
- Refer: Spring3MVC\_Hibernate\_CRUD.zip
  - Illustrates integrating with Hibernate API
  - Illustrates using @ModelAttribute to map form request parameters to entity class
  - Illustrates using Spring validation framework: Validator interface, BindingResult interface to validate form data



### ● Activity 4

- Modify Activity 3 to validate employee

- Employee should be stored in the map data of EmployeeController only if valid data is entered.
- Validation Rules:
  - Employee ID is required and should be numeric
  - Employee Name is required and minimum 3 digits
  - Salary is required and should be greater than zero.



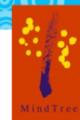
### ● PropertyEditors

- The `java.beans.PropertyEditor` interface provides a means to customize how String values are mapped to non-String types.
- `java.beans.PropertyEditorSupport` is a support class to help build property editors.
- Some important methods which has to be overridden in our `PropertyEditor` class are listed below:

Method	Description
<code>void setAsText(String text)</code>	Sets the property value by parsing a given String
<code>String getAsText()</code>	Gets the property value as a string suitable for presentation to a human to edit
<code>Object getAsValue()</code>	Gets the value of the property.

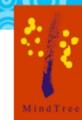
Refer

:<http://download.oracle.com/javase/1.5.0/docs/api/java/beans/PropertyEditorSupport.html>



- Spring Framework comes with several custom editors based on `PropertyEditorSupport`.
  - For Example “CustomDateEditor” is used to set a `java.util.Date` property from a String using a custom `java.text.DateFormat` object.
- In a web application data entered from form fields are of type `String`, If type conversion has to happen between the entered date in the `String` format to a `java.util.Date`, you need to register explicitly using `WebDataBinder`.
- Annotating controller methods with `@InitBinder` allows you to configure web data binding directly within your controller class

```
@InitBinder  
public void initBinder(WebDataBinder binder) {  
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");  
    binder.registerCustomEditor(Date.class, new CustomDateEditor(  
        dateFormat, true));  
}
```



## Type Conversion

- Writing your own PropertyEditors
  - Converts data entered in the form House, Street, City to Address property

```
/*
 * @author Banu Prakash
 * © 2011 MindTree Limited
 *
 */
public class AddressEditor extends PropertyEditorSupport {
    private String[] strAddressData;
    private Address address;
    /* (non-Javadoc)
     * @see java.beans.PropertyEditorSupport#setAsText(java.lang.String)
     */
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
        if( text != null) {
            strAddressData = text.split(",");
            if( strAddressData.length != 3) {
                throw new IllegalArgumentException("Address should have House No,Street,City");
            }
        } else {
            throw new IllegalArgumentException("Address should have House No,Street,City");
        }
    }
    /* (non-Javadoc)
     * @Override
     */
    public Object getValue() {
        return new Address(strAddressData[0],strAddressData[1],strAddressData[2]);
    }
}
```

CONFIDENTIAL For limited circulation only

© 2011 MindTree Limited

Slide 46

Data entered as “house,street,city” will be passed as argument to setAsText(String text). The value returned from getValue() will be assigned to property of type Address

### Example

```
public class Customer {
    private String name;
    private Address address;
}
```

```
<bean id="customer" class="com.mindtree.entity.Customer">
    <property name="name" value="Banu Prakash" />
    <property name="address" value="No 5,Yelahanka new Town,
Bangalore" />
</bean>
```



## Activity

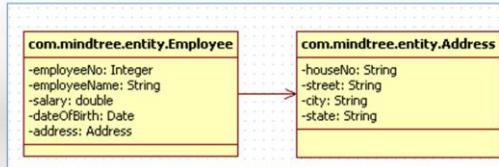
### ● Activity 5

#### ● Modify Activity 4.

- The user interface should accept dateOfBirth and address. These data entered in form field are in the form of String data type.
- Using WebDataBinder convert the data entered in form field for Date Of Birth to Date and data entered in address field to Address respectively.
- Date is entered as dd-MMM-yyyy
  - Example: 10-JAN-1987
- Address is entered in the form houseNo, street, city, state
  - Example: No: 5, I A Main Road, Yelahanka New town, Bangalore, Karnataka.

empAdd.jsp

Employee ID	<input type="text"/>
Name	<input type="text"/>
Salary	<input type="text"/>
Date of Birth	<input type="text"/>
Address	<input type="text"/>
<input type="button" value="Submit"/>	



CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 47



## Pre-populating form fields

### ● Pre-populate Form Fields

- @ModelAttribute annotated methods will be executed *before* the chosen @RequestMapping annotated handler method.

The screenshot shows a web page at <http://localhost:8080/vehicleRental/addVehicle.action>. The form includes fields for Registration No., Category (with options Car, Truck, Bus, UU), Manufacturer (with options Ford, Toyota, Honda), Mileage, Daily Rent, and Fuel Type (Petrol, Diesel, Hybrid). A callout from the 'Category' dropdown points to the `@ModelAttribute("categoryList")` annotation in the `populateCategory()` method of the `VehicleController`.

```
@Controller
public class VehicleController {

    @Autowired
    private VehicleRentalService rentalService;

    @Autowired
    private VehicleValidator validator;

    @ModelAttribute("categoryList")
    public Map<String, String> populateCategory(){
        Map<String, String> catMap = new LinkedHashMap<String, String>();
        catMap.put("Car", "Car");
        catMap.put("Truck", "Truck");
        catMap.put("Bus", "Bus");
        return catMap;
    }

    @RequestMapping(value="/addVehicle.action",
                    method = RequestMethod.GET)
    public String getVehicleAddPage(Model model){
        Vehicle vehicle = new Vehicle();
        model.addAttribute("vehicle", vehicle);
        return "addVehicle"; // InternalResourceViewResolver sends JSP
    }
}
```

`<tr>
 <td>Category :</td>
 <td><form:select path="category">
 <form:option value="--SELECT--" />
 <form:options items="${categoryList}" />
 </form:select>
 </td>
</tr>`

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited

Slide 48

For the following Request "<http://localhost:8080/VehicleRental/addVehicle.action>"  
VehicleController's GET @RequestMapping(value="/addVehicle.action",  
method=RequestMethod.GET) will be mapped. But before this method is invoked, it  
invokes all the methods annotated with @ModelAttribute for pre-populating the form  
fields.

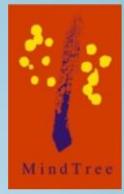
## Example



### ● Code Example:

- Spring3MVC\_Vehicle\_Rental\_App.zip
  - A Spring MVC web application which illustrates integration with Hibernate
  - Illustrates using Validators, ResourceMapping and populating form fields
  - Illustrates implementing Open Session In View Filter design pattern.

- Add a view to display all Categories and vehicles belonging to that category
  - Note: Do not use lazy loading, absorb the difference with and without OpenSessionInViewFilter configuration.
- Add the following Validation
  - While renting a vehicle “Returned Date” should not be before the “Booked Date”.



## References

Contains the reference that will supplement the self learning and will be needed for completing the assignments & practice questions

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited



## References

- Spring MVC Documentation:
  - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html>
  - <http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/orm/hibernate3/support/OpenSessionInViewFilter.html>
- Spring MVC Tutorial
  - <http://www.mkyong.com/tutorials/spring-mvc-tutorials/>
- Spring Form tags
  - <http://static.springsource.org/spring/docs/2.0.x/reference/spring-form.tld.html>
  - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/view.html>
  - <http://www.vaannila.com/spring/spring-form-tags-1.html>
- Spring Samples:
  - <https://src.springframework.org/svn/spring-samples/>
  - Refer: [mvc-basic/](#) [mvc-showcase/](#) [petcare/](#) [jpetstore/](#) [mvc-ajax/](#)



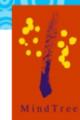
## Explore More!!

Never let your curiosity die!

CONFIDENTIAL: For limited circulation only

©2011 MindTree Limited

## Explore more



- Session handling

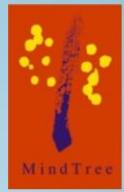
- <http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/web/bind/annotation/SessionAttributes.html>
- <http://www.infoq.com/articles/spring-2.5-ii-spring-mvc>
- <http://static.springsource.org/spring/docs/2.5.x/reference/mvc.html>

- Spring's multipart (file upload) support

- <http://static.springsource.org/spring/docs/2.5.x/reference/mvc.html>

- Internationalization

- <http://www.mkyong.com/spring-mvc/spring-mvc-internationalization-example/>



**Our Mission**

**Successful Customers**  
**Happy People**  
**Innovative Solutions**

Contact Person

Contact\_contact@mindtree.com  
+Country code-Phone  
[www.mindtree.com](http://www.mindtree.com)

CONFIDENTIAL: For limited circulation only

© 2011 MindTree Limited