Mindtree
Welcome to possible

Spring MVC
Validators, Convertors and i18N

© Mindtree limited 2012

Objectives

- Enhance the web application using validators, Type conversion and formatting, i18(internationalization),themes, templates

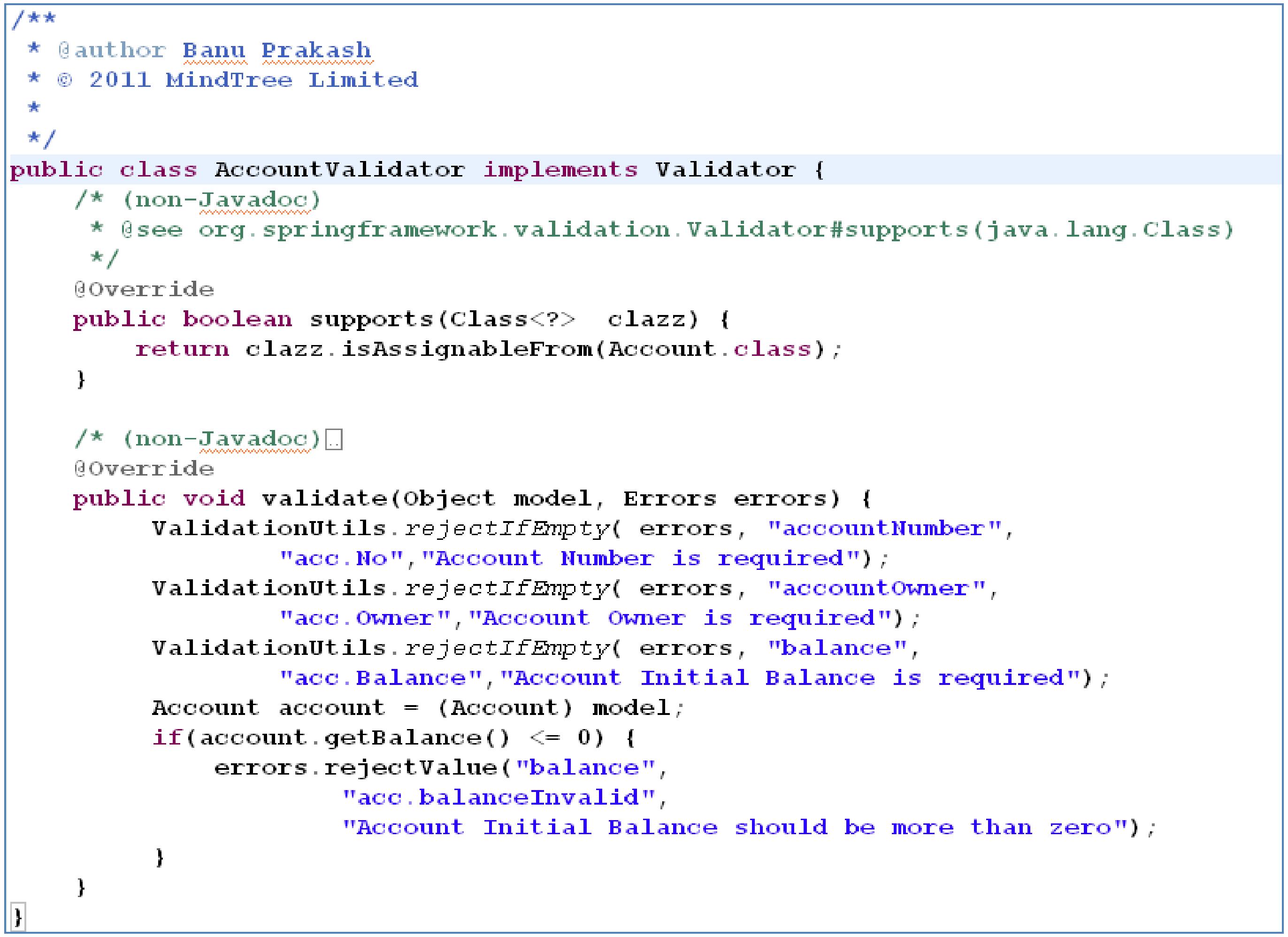
Validation

- **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



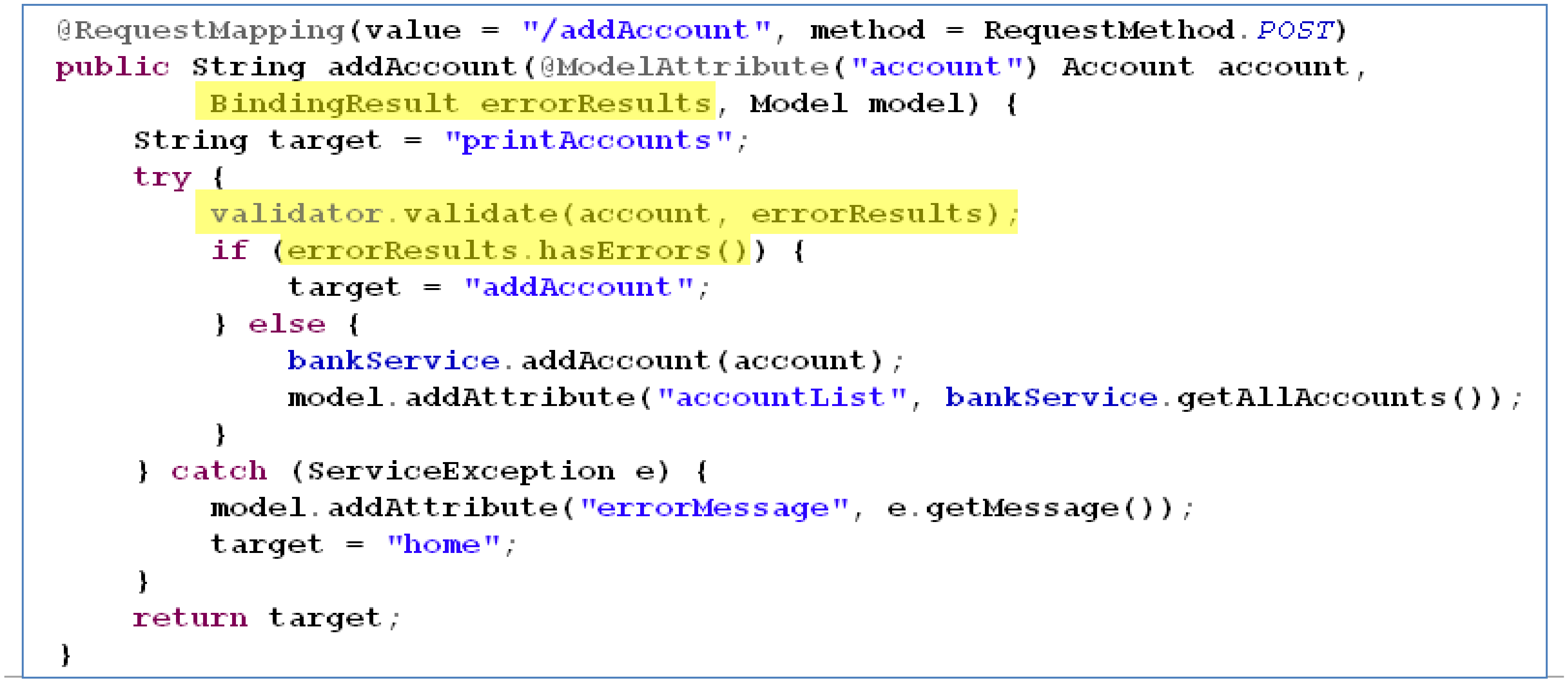
```
/**
 * @author Banu Prakash
 * © 2011 MindTree Limited
 *
 */
public class AccountValidator implements Validator {
    /* (non-Javadoc)
     * @see org.springframework.validation.Validator#supports(java.lang.Class)
     */
    @Override
    public boolean supports(Class<?>  clazz) {
        return clazz.isAssignableFrom(Account.class);
    }

    /* (non-Javadoc)
    @Override
    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");
        }
    }
}
```

Mindtree

Validation

- 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;
}
```

Mindtree

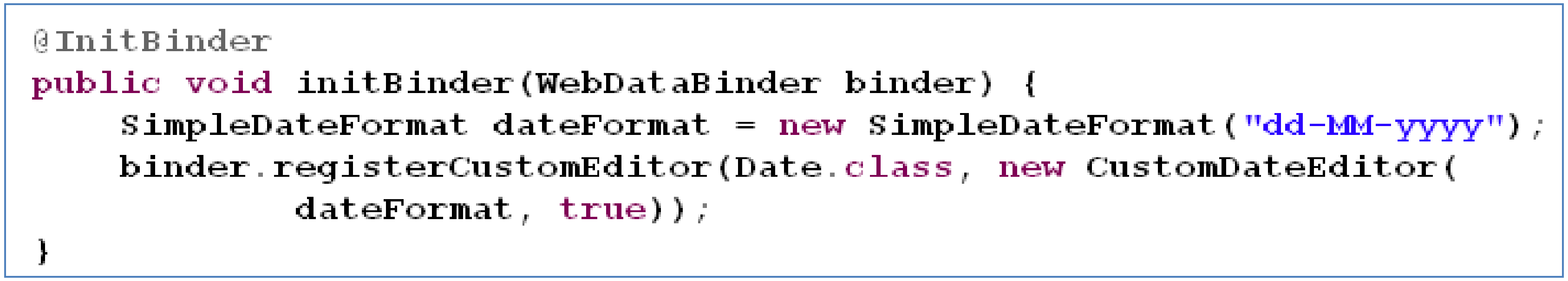
Type Conversion

- **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
void setAsText(String text)	Sets the property value by parsing a given String
String getAsText()	Gets the property value as a string suitable for presentation to a human to edit
Object getAsValue()	Gets the value of the property.

Type Conversion

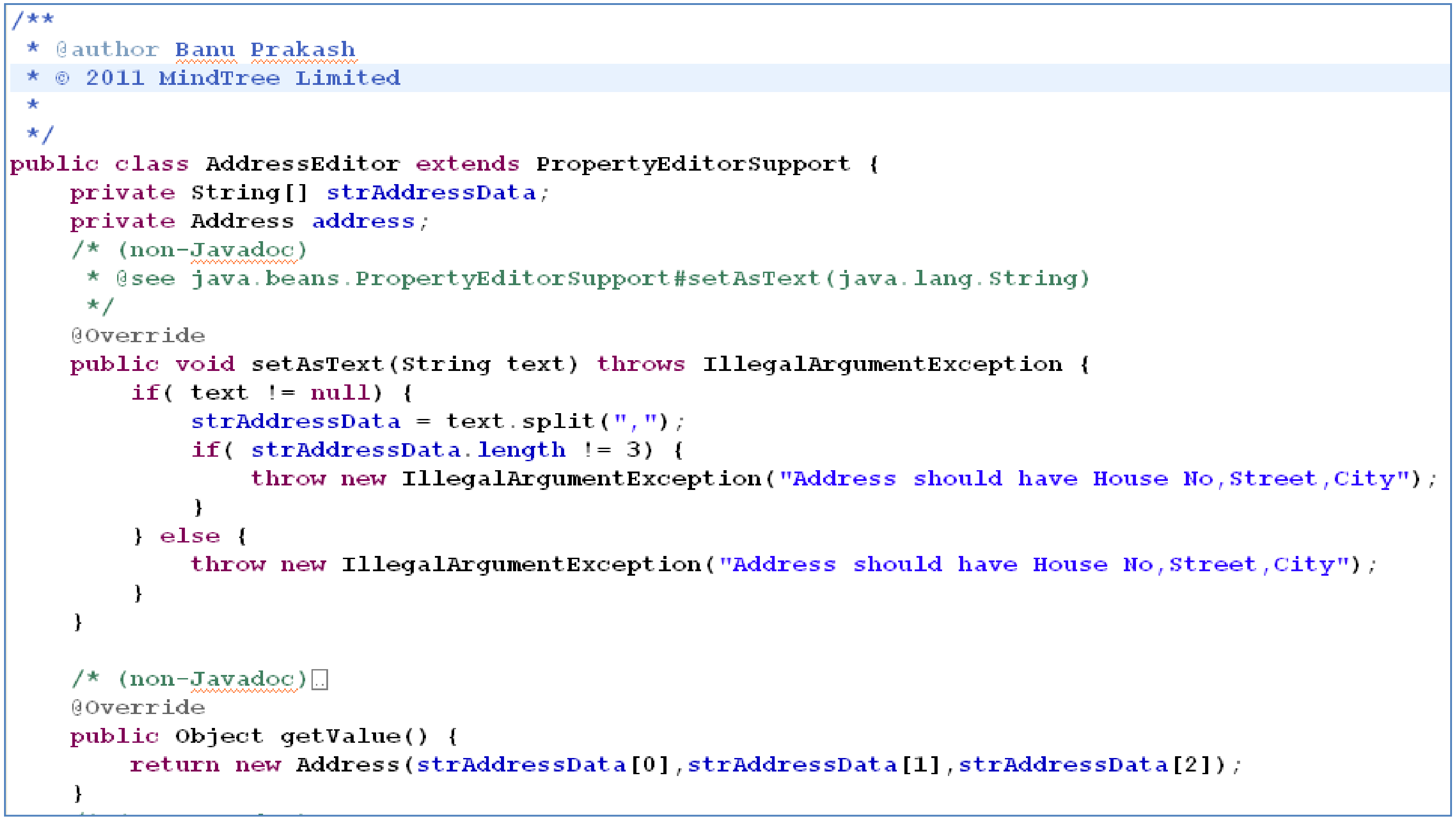
- Spring Framework comes with several custom editors based on PropertyEditorSupport.
  - For Example “CustomDateEditor” is used to set a java.util.Date property from a String using a custom java.text.DateFormat object.
- **In a web application data entered from form fields are of type String, If type 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



```
@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



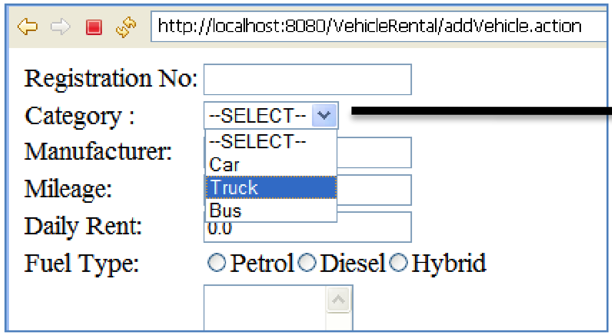
```
/**
 * @author Banu Prakash
 * © 2011 MindTree Limited
 *
 */
public class AddressEditor extends PropertyEditorSupport {
    private String[] strAddressData;
    private Address address;
    /* (non-Javadoc)
     * @see java.beans.PropertyEditorSupport#setAsText(java.lang.String)
     */
    @Override
    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]);
    }
```

Mindtree

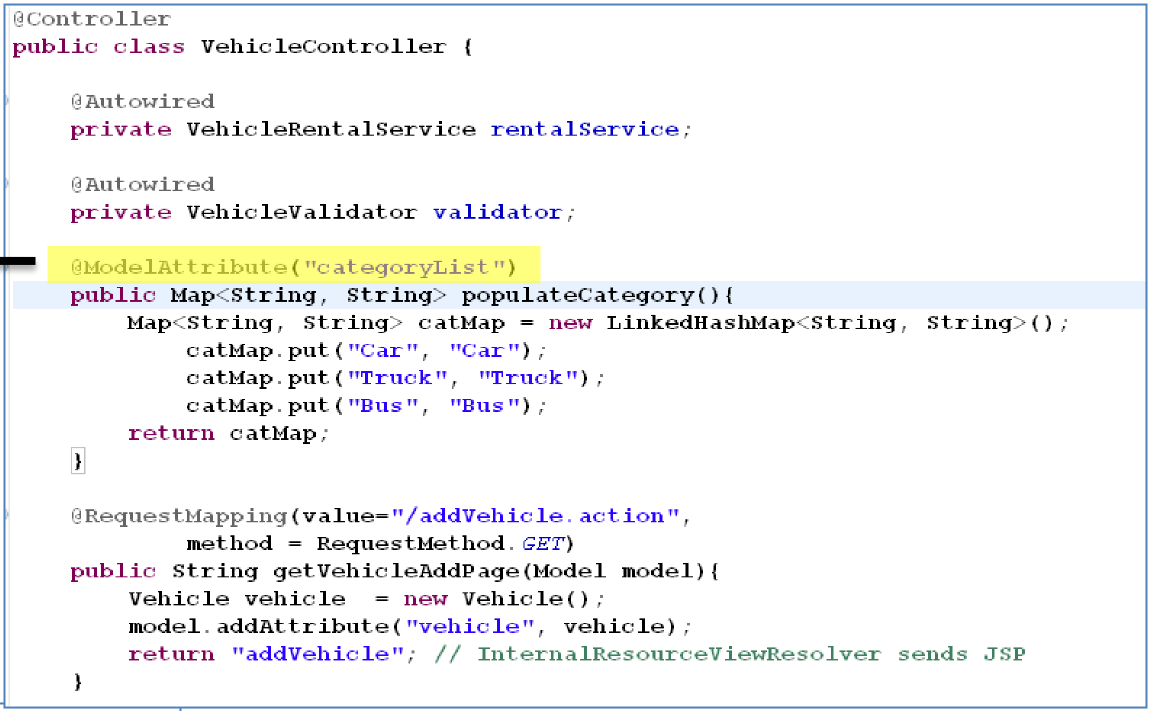
Pre-populating form fields

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



http://localhost:8080/VehicleRental/addVehicle.action

Registration No:	
Category :	--SELECT-- (--SELECT--, Car, Truck, Bus)
Manufacturer:	
Mileage:	
Daily Rent:	0.0
Fuel Type:	○ Petrol ○ Diesel ○ Hybrid



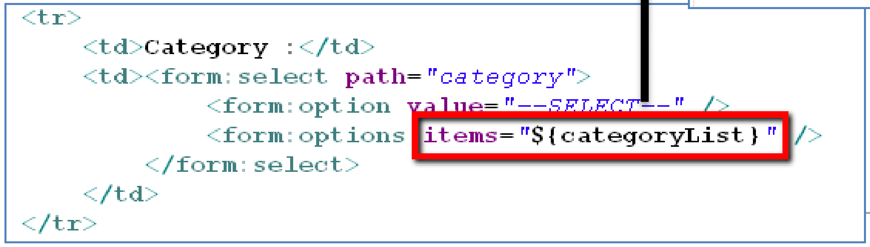
```
@Controller
public class VehicleController {

    @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>
```

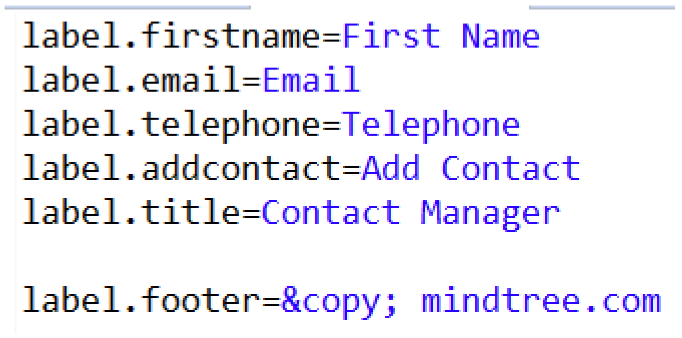
Mindtree

Internationalization & Localization

- Internationalization and localization are means of adapting computer software to different languages and regional differences.
- Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes.
- Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text
- The terms are frequently abbreviated to the numerous i18n (where 18 stands for the number of letters between the first i and last n in internationalization) and L10n respectively, due to the length of the words.

Spring MVC Internationalization (i18n)

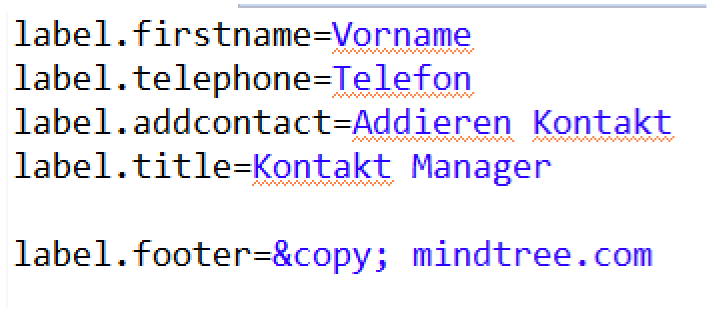
- Message Resources File
  - Create two files messages.properties [default]



```
label.firstname=First Name
label.email=Email
label.telephone=Telephone
label.addcontact=Add Contact
label.title=Contact Manager

label.footer=&copy; mindtree.com
```

  - and messages_de.properties[German]

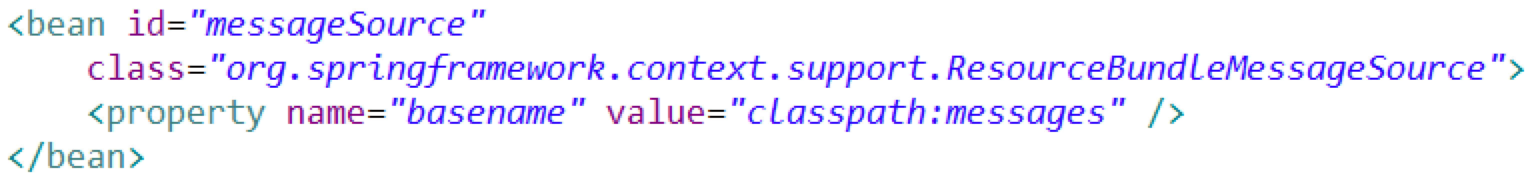


```
label.firstname=Vorname
label.telephone=Telefon
label.addcontact=Addieren Kontakt
label.title=Kontakt Manager

label.footer=&copy; mindtree.com
```

Spring MVC configuration

- Configure ResourceBundleMessageSource in the following way.
- The attribute to be noted here is the basename. This tells the Spring where to look for the properties file when a request for localized text comes in.
- We tell the Spring framework to look for file named messages.properties file in the classpath. It is called as basename because, whenever the request for a localized string comes, the locale or language code will be automatically appended to the basename value. So if the french string is requested, then messages_fr will be automatically picked from the classpath



```xml
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="classpath:messages" />
</bean>
```

Mindtree

Spring MVC configuration

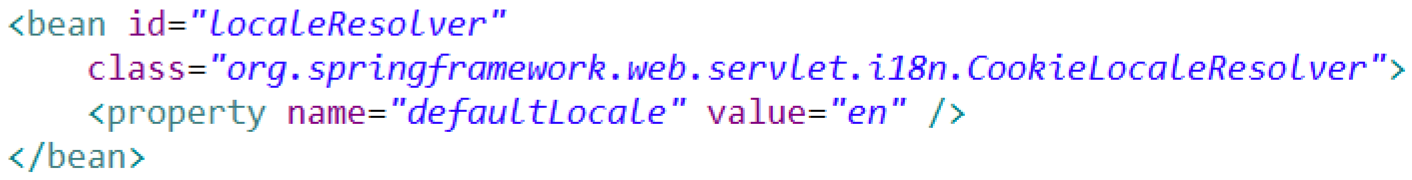
- LocaleChangeInterceptor : This interceptor allows for changing the current locale on every request, via a configurable request parameter. Usually this parameter is lang. So, anytime a user send a request with request parameter “lang=en”, his locale automatically be set as English.

```
<bean id="localeChangeInterceptor"
    class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
    <property name="paramName" value="lang" />
</bean>
```

Mindtree

Spring MVC configuration

- **CookieLocaleResolver:**
  - It’s a LocaleResolver implementation that uses a cookie sent back to the user in case of a custom setting.
  - Remember that sending locale or lang parameter in request is actually a custom setting.
  - The cookie sent to user is used in further communication as user will not have to set the locale in every request.
  - If this cookie has been generated, each request will have this overridden locale in request.



```
<bean id="localeResolver"
    class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
    <property name="defaultLocale" value="en" />
</bean>
```

Mindtree

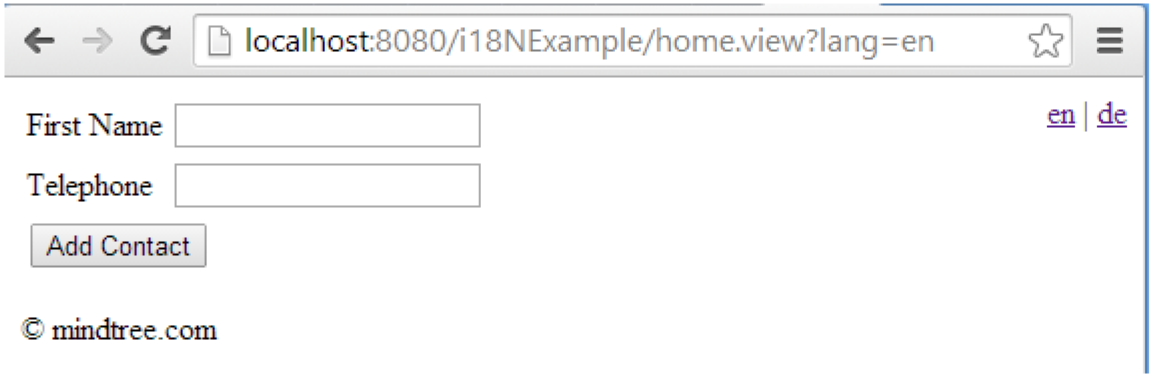
Spring MVC configuration

- In JSP pages use the **spring:message** to display the message from the corresponds properties file by checking the current user’s locale.

```
<table>
    <tr>
        <td><form:label path="firstname">
                <spring:message code="label.firstname" />
            </form:label>
        </td>
        <td><form:input path="firstname" /></td>
    </tr>
    <tr>
        <td><form:label path="telephone">
                <spring:message code="label.telephone" />
            </form:label></td>
        <td><form:input path="telephone" /></td>
    </tr>
    <tr>
        <td colspan="2">
            <input type="submit"
                value="<spring:message code="label.addcontact"/>" />
        </td>
    </tr>
</table>
```

Spring MVC configuration

- View for English locale setting



localhost:8080/i18NExample/home.view?lang=en

en | de

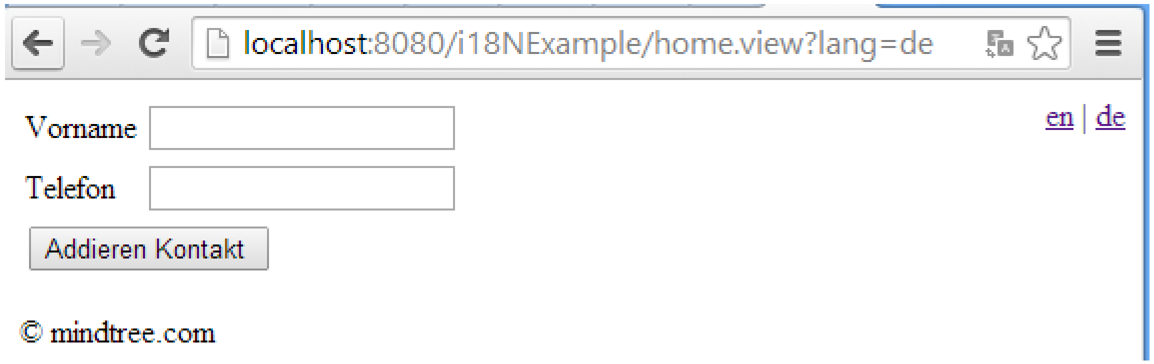
First Name

Telephone

Add Contact

© mindtree.com

- View for German locale setting



localhost:8080/i18NExample/home.view?lang=de

en | de

Vorname

Telefon

Addieren Kontakt

© mindtree.com

Welcome to possible

India | USA | UK | Germany | Sweden | Belgium | France | Switzerland | UAE | Singapore | Australia | Japan | China