

# MODULE 9

# INTERNATIONALIZATION

# TABLE OF CONTENT

- What is Internationalization?
- Language Property Files
- Configuration for Internationalization
- Using Language Files in Web Pages
- Access to Translated Message in Java
- Translation of Error Messages
- Locale Chosen by the User
- Internationalization of Dynamic Data

# What is Internationalization?

- ▶ To build localized applications
- ▶ To display texts according to the language chosen by the user
- ▶ Static labels
  - Using bundles
    - **ResourceBundleMessageSource**
- ▶ Dynamic values
  - From **databases**

# Language Property Files

- ▶ Bundle of property files
- ▶ One property file by language
  - Containing values for static labels translated in this language
- ▶ Name of file
  - **rootName\_XX.properties**
    - where **rootName** is the same for all files
    - and **XX** are two characters corresponding to the language  
**fr, en, nl, de, ...**
  - E.g.,
    - *general\_en.properties*
    - *general\_fr.properties*
    - *general\_nl.properties*
    - + *general.properties* (default)

# Language Property Files

- ▶ Each file contains **key-value** pairs
  - **Key** : identifier of the label
  - **Value** : translation of the label in the language
- ▶ *E.g.*,
  - general\_fr.properties:

*firstName=Prénom*

*lastName=Nom de famille*

*welcome=Bienvenue !*

*age=Quel est ton âge?*

*sendButton=Envoyer*

- general\_en.properties

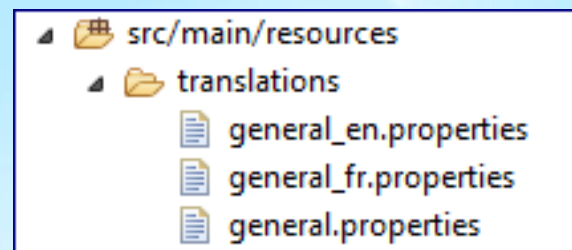
*firstName=First Name*

*lastName=Last Name*

*welcome=Welcome !*

*age=How old are you?*

*sendButton=Send*



# Configuration for Internationalization

- In Configuration class add 2 bean declarations
  - **DefaultMessageCodesResolver**
  - **ResourceBundleMessageSource**

```
@Bean
public DefaultMessageCodesResolver defaultMessageCodesResolver() {
    DefaultMessageCodesResolver defaultMessageCodesResolver = new DefaultMessageCodesResolver();
    return defaultMessageCodesResolver;
}

@Bean
public ResourceBundleMessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    messageSource.setDefaultEncoding("UTF-8");
    messageSource.setBasenames("translations/general", "translations/errors");
    messageSource.setUseCodeAsDefaultMessage(true);
    return messageSource;
}
```

Package of the language property files

Root name of files

For error messages translations

# Using Language Files in Web Pages

- ▶ **<spring:message>**
  - To access to the translated message
  - Attribute
    - **code**
      - Value of the key in language property file
- ▶ E.g,

```
<form:label path="firstName">  
    <spring:message code="firstName"/>  
</form:label>
```



# Access to Translated Message in Java

- ▶ Access to **ResourceBundleMessageSource** through injection
  - Use **getMessage** method with key and locale
  - E.g,

```
@Controller
@RequestMapping(value="/testForm")
public class TestFormController {

    @Autowired
    private MessageSource messageSource;

    ...

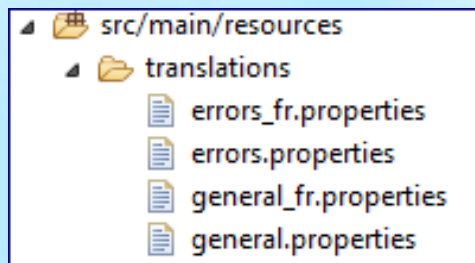
    @RequestMapping(method=RequestMethod.POST)
    public String getFormData(Model model,
                              @Valid @ModelAttribute(value="inscriptionForm") Form inscriptionForm,
                              Locale locale)
    { model.addAttribute("messageToDisplay", messageSource.getMessage("welcome", null, locale));
      return "integrated:form";
    }
}
```

↓  
Key in language property file



# Translation of Error Messages

- ▶ Add language property files for error messages



- ▶ Key in language property files
  - Use NotNull, ...
  - + either a class name or an attribute of the model

# Translation of Error Messages

► E.g,

```
<form:form id="inscription" method="POST"
  action="/first/form/userInscription"
  modelAttribute="userForm">
  <form:label path="name"> Name </form:label>
  <form:input path="name"/>
  <form:errors path="name"/>
  <br>
  <form:label path="zipCode"> Zip Code </form:label>
  <form:input path="zipCode"/>
  <form:errors path="zipCode"/>
  <br>
  <form:button>Submit</form:button>
  <br>
  <c:if test="${not empty message}"> ${message} </c:if>
</form:form>
```

# Translation of Error Messages

- ▶ errors.properties

```
NotNull.java.lang.String=The string may not be null  
NotNull.userForm.zipCode=The zipcode may not be null
```

- ▶ errors\_fr.properties

```
NotNull.java.lang.String=La chaîne de caractères ne peut pas être vide  
NotNull.userForm.zipCode=Le code postal ne peut pas être vide
```

# Locale Chosen by the User

- ▶ The user can choose the language
  - E.g, by a clic on a icon
    - Usually representing flags

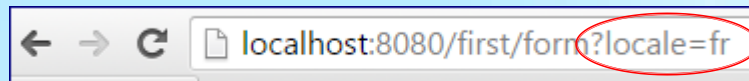
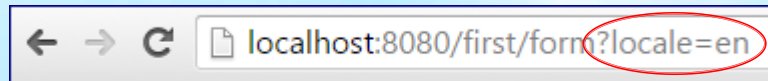


# Locale Chosen by the User

## ► How to do?

- Add a locale parameter to each url. E.g, ...?locale=fr
  - See definition of button in jsp pages
- Create a session cookie
  - To store the chosen locale
  - In Configuration class
    - Use *CookieLocaleResolver*
- Use an interceptor to intercept request before call of controllers
  - That catches the locale parameter and places it in the session cookie
  - In Configuration class
    - Use *LocaleChangeInterceptor*

# Locale Chosen by the User – *jsp* page



- Use **<spring:url>** to define url
  - Use **<spring:param>** to add parameter to url

# Locale Chosen by the User – *jsp* page

► E.g,

```
<spring:url var="localeFr" value="">
    <spring:param name="locale" value="fr" />
</spring:url>

<spring:url var="localeEn" value="">
    <spring:param name="locale" value="en" />
</spring:url>

<a href="{localeFr}"><img .../></a>
<a href="{localeEn}"><img.../></a>
```

→ Add "?locale=fr" to url

→ Add "?locale=en" to url

→ Definition of buttons using spring:url



# Locale Chosen by the User – Configuration

## ► In Configuration class

```
@Configuration
public class MainConfig extends WebMvcConfigurerAdapter {

    @Bean
    public LocaleResolver localeResolver() {
        CookieLocaleResolver resolver = new CookieLocaleResolver();
        resolver.setDefaultLocale(new Locale("fr"));  → Default locale
        resolver.setCookieName("myLocaleCookie");
        resolver.setCookieMaxAge(-1);  → -1 for session cookie
        return resolver;    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        LocaleChangeInterceptor interceptor = new LocaleChangeInterceptor();
        interceptor.setParamName("locale");  → Name of parameter to intercept in url
        registry.addInterceptor(interceptor);    }
}
```

# Internationalization of Dynamic Data

- ▶ Translation of labels in database
- ▶ E.g,

