Web Component Development Using Java

Session: 14

Internationalization

Objectives

❖ Explain the concept of internationalization

❖ Explain the concept of localization

❖ Explain the role of Unicode character set in internationalization

❖ Explain the resource bundling mechanism and resource bundle for various locales

❖ Explain how to format dates in servlets for internationalization

❖ Explain how to format currency in servlets for internationalization

❖ Explain how to format numbers in servlets for internationalization

❖ Explain how to format percentages in servlets for internationalization

❖ Explain how to format messages in servlets for internationalization

❖ Explain various tags available in the JSTL internationalization tag library

❖ Explain how to format dates and currencies using JSTL I18N tags

❖ Explain how to format percentages and messages using JSTL I18N tags

Internationalization

❖ Is the method of designing an application that can be adapted to a region or a language without much change in the technology.

❖ Is used in creating internationalized Web applications that standardize formatted numeric and date-time output.

Localization

❖ Localization is a process of making a product or service language, culture, and local 'look-and-feel' specific.

❖ Localizing a product does not only mean a language translation, but also formatting of time, currencies, messages, and dates.

❖ In Java:
- ❑ A `Locale` is a simple object, which identifies a specific language and a geographic region.

- ❑ To create international Java applications, the use of `java.util.Locale` class is a must.

- ❑ Locales are used in entire Java class libraries data for formatting and customization.

Unicode

❖ Is a coding system that has codes for all the major language of the world.

❖ Is a 16-bit character encoding.

❖ Uses UCS-2- a fixed-width two byte encoding that simply encodes each code point from U+0000 to U+FFFF as itself.

❖ Allows Java to handle international characters for most of the languages of the world.

❖ Figure depicts the symbol table supported in Unicode.
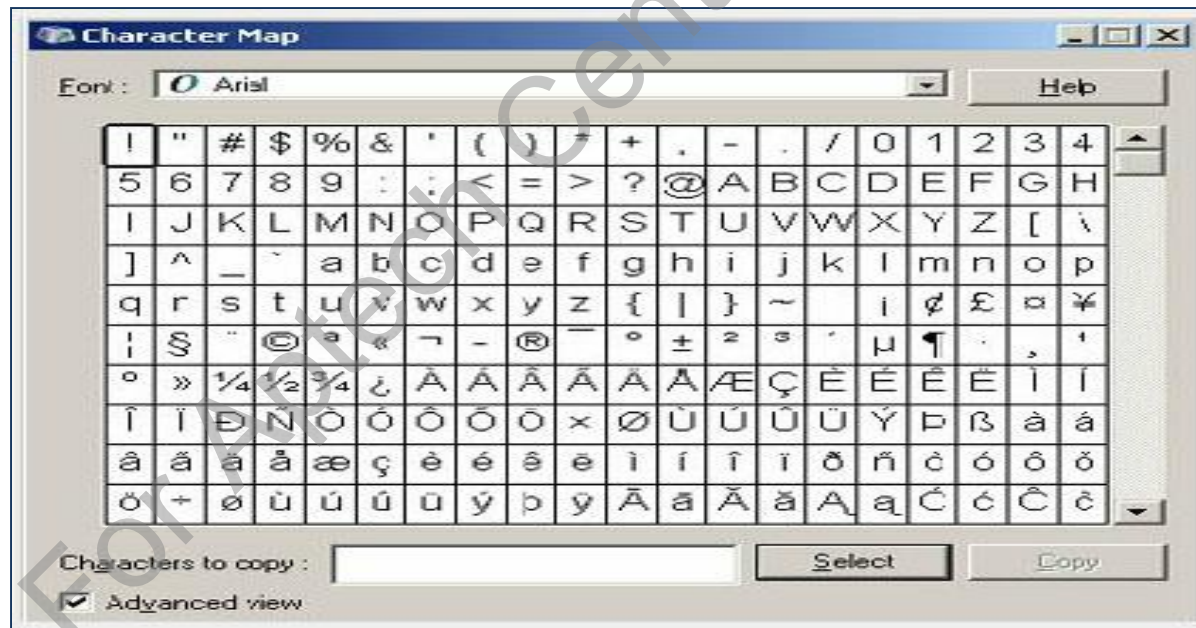
Locale

❖ Locale represent specific geographical, political, or cultural region.

❖ Locale is denoted by the standard format **xx_YY**, where, xx stands for two-letter language code in lower case, and **YY** stands for two-letter country code in upper case.

❖ Some of the examples of locales are zh_CN for Shanghai, China; ko_KR for Seoul, Korea; and en_US for English, US.

❖ Figure depicts the use of various locales.
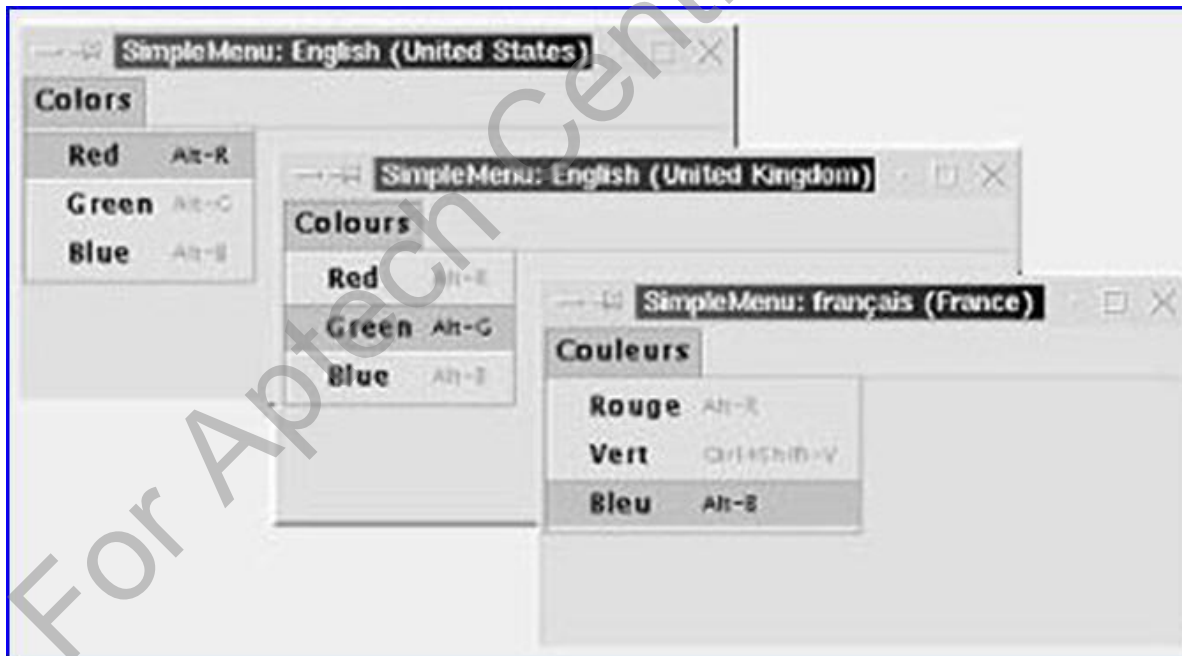
Creating a ResourceBundle 1-2

How to create a internalization of Web application in Java?

ANSWERS

Create text files with key/value pairs representing text from different locales and store them under WEB-INF folder.

❖ For example, the key "**Welcome**" is associated with a value or message in different `ResourceBundle` property files.

  ❑ Key – Represents the word.

  ❑ Value – Represents the translation in the specific locale.

Creating a ResourceBundle 2-2

❖ The code snippet shows the `ResourceBundle` property file named `DemoResources_es_ES.properties` containing **Spanish** equivalent for the key "**Welcome**".

```
#Spanish language resources
Welcome = Hola y recepción
```

❖ The code snippet shows the `ResourceBundle` property file named `DemoResources_en_US.properties` containing **United States English** equivalent for the key "**Welcome**".

```
#English language resources
Welcome = Hello and welcome
```

❖ Internationalization of an application can be achieved with the help of resource bundles.

❖ **Resource Bundle**:

❑ Is a set of related classes that inherit from the `ResourceBundle`.

❑ The subclass of `ResourceBundle` has the same base name with an additional component that identifies locales.

❑ For example, if resource bundle is named `DemoResources` and along with it, locale-specific classes can be related as in `DemoResources_en_US`.

❑ Helps to build server-side code that gives output based on the location and language of the user.

❑ Makes the job easier by avoiding the writing of multiple version of a class for different locales.

❖ There are several methods in `ResourceBundle` class.

```
public static final ResourceBundle getBundle(String
baseName)
```

- ❑ In order to get resource bundle for getting the locale-specific data, the `getBundle()` method is used.
- ❑ This method gets a resource bundle using the specified base name, the default locale, and the class loader of the caller.

❖ The code snippet demonstrates the method to get a resource bundle using the specified base name and locale.

```
/* This snippet creates ResourceBundle by invoking
getBundle() method, specifying the base name */

ResourceBundle labels =
ResourceBundle.getBundle("DemoLabelsBundle");
```

public abstract Enumeration getKeys()

❑ Returns an enumeration of the keys present in the property file.

❖ The code snippet shows the use of the `getKeys()` method.

```
// this snippet displays the value of the keys

static void iterateKeys(Locale currentLocale) {

 ResourceBundle labels =
ResourceBundle.getBundle("LabelsBundle",currentLocale)
;

  Enumeration bundleKeys = labels.getKeys();

 while (bundleKeys.hasMoreElements()) {
     String key = (String)bundleKeys.nextElement();
     String value = labels.getString(key);
     System.out.println("key = " + key + ", " +
         "value = " + value);
 }
}
```

Internationalizing Servlets 4-6

public Locale getLocale()

❑ Returns the locale of this resource bundle.

❑ Can be used after the calling of `getBundle()` to check whether the returned resource bundle really corresponds to the requested locale or not.

❑ Returns the locale for the current resource bundle.

❖ The code snippet shows the use of the `getLocale()` method.

```
// This snippet gets the user's Locale

    Locale locale = request.getLocale();

ResourceBundle bundle =
ResourceBundle.getBundle("i18n.WelcomeDemoBundle"
, locale);

String welcome = bundle.getString("DemoWelcome");
```

Internationalizing Servlets 5-6

```
public final Object getObject(String key)
```

- Gets an object for the given key from the resource bundle.

- Gets the object from the resource bundle and if it fails, the parent resource bundle is called using parent's `getObject()` method.

- Throws a `MissingResourceException` if it fails.

- For example, `int[] myDemoIntegers = (int[]) myDemoResources.getObject ("intList");`

Internationalizing Servlets 6-6

`public final String getString(String key)`

❑ Returns a string for the given key from the resource bundle or one of its parents.

❖ The code snippet shows the use of `getString()` method.

```
/** Retrieves the translated value from the
ResourceBundle by invoking the getString method as
follows **/

String value = labels.getString(key);
```

Formatting in Servlets

❖ The formatting of the numbers, currency, date, and percentage helps the programmer to format these values based on the locale or the region of the user.

Formatting Numbers

Formatting Currency

Formatting Percentages

Formatting Dates

Formatting in Servlets

Formatting Messages
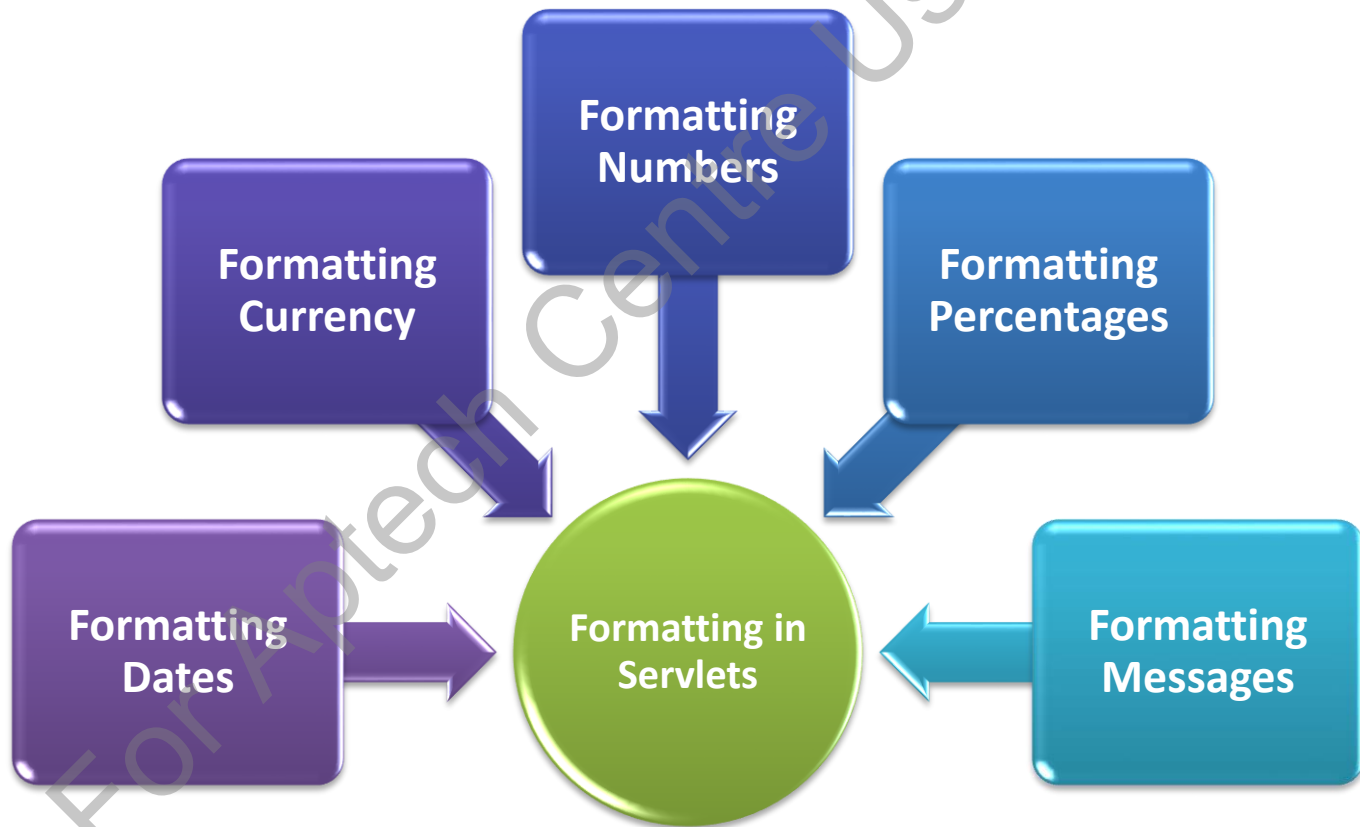
Formatting Dates 1-4

❖ The formats in which dates can be displayed include **Predefined Formats** and **Customizing Formats.**

Predefined Formats

❖ The `DateFormat` style is predefined and locale-specific and is easy to use. The styles are as follows:

- ❑ **SHORT** - it is completely numeric, such as `11.14.50` or `3:30pm`.
- ❑ **MEDIUM** - it is longer, such as `Jan 10, 1954`.
- ❑ **LONG** - it is longer, such as `January 10, 1954` or `3:50:32pm`.
- ❑ **FULL** - it is completely specified, such as `Tuesday, April 14, 1954 AD` or `3:50:42pm PST`.

❖ The two steps in formatting of date using the `DateFormat` class are as follows:

Step 1
- Creating a formatter with the `getDateInstance()` method

Step 2
- Invoking the `format()` method, which returns formatted date in the form of string

Formatting Dates 2-4

Customizing Formats

❖ Most of the time, the predefined formats are enough, but sometimes customized format is required.

❖ To achieve custom formatting, the `SimpleDateFormat` class is used.

❖ `SimpleDateFormat` class:

❑ Formats and parses dates in a locale-sensitive manner.

❑ The code snippet demonstrates the use of the `SimpleDateFormat` class.

```
/* this snippet formats the date as per the
specified style */

SimpleDateFormat fmt = new SimpleDateFormat("yyyy-
MM-dd", Locale.US);
```

Formatting Dates 3-4

❖ The several classes for formatting dates are as follows:

- ❑ **DateFormat** - Formats and parses the dates and time in a language independent manner.

- ❑ **DateFormat.Field** - Is a nested class that is used as attribute keys in the `AttributedCharacterIterator`.

- ❑ **DateFormatSymbol** - Encapsulates localizable date-time formatting data, such as names of months, days of week, and the time zone data.

- ❑ **DateFormatter** - Formats the date as per the format of the current locale.

❖ Figure depicts the various date formats.

Formatting Currency 1-4

❖ The formatting of the currency is necessary, so that the users can be benefited by the availability of the locale specific formatting.

❖ Conversion of one currency to another requires some conversion in value.

❖ The formatting of currency value is much more involved compared to formatting date and time.

❖ Some of the classes used for formatting currency are as follows:

`Currency`	• This class represents currency. The currencies are identified by ISO 4217 currency codes.
`NumberFormat`	• It provides methods to determine the number formats of the locales and their name. It helps to format and parse numbers for any locales.

Formatting Currency 2-4

❖ Some of the methods of currency class are as follows:

public String getCurrencyCode()

❑ This method gets the currency code of the currency as per the ISO 4217 codes.

❑ The code snippet shows the use of the getCurrencyCode() method.

```
/* this snippet returns the currency code of the
currency as per the ISO 4217 codes */

public String getCurrencyCode(){
    return currency.getCurrencyCode();
}
```

Formatting Currency 3-4

`public String getSymbol()`

❑ This method gets the currency symbol for the default locale.

❑ The code snippet shows the use of `getSymbol()` method.

```
/* this snippet gets the currency symbol for
default locale */


public String getCurrencySymbol(){

    return currency.getSymbol();

}
```

public static Currency getInstance(Locale locale)

❑ This method returns the `Currency` instance for the country of the specified locale.

❑ The code snippet shows the use of `getInstance()` method.

```java
// this snippet gets the currency instance //
import java.text.NumberFormat;
import java.util.Currency;


public class CurrencyClass {
    public static void main ( String args []){


// returns locale-specific currency instance
Currency localeCurrency =
Currency.getInstance(locale);
    }
}
```

Formatting Numbers 1-5

❖ Some of the methods of `NumberFormat` class are as follows:

```
public final String format(double number)
```

❑ This method is the specialization of the format.

❑ The code snippet shows the use of `format()` method.

```
/* getPercentInstance() returns a percentage
format for the current default locale */

NumberFormat nft =
NumberFormat.getPercentInstance(locale);

String formatted = nft.format(0.51);
```

public Currency getCurrency()

❑ This method provides the number format while formatting currency values.

❑ The value derived initially is locale dependent.

❑ If no valid currency is determined and no currency has been set using `setCurrency` the returned value may be null.

❑ The code snippet shows the use of `getCurrency()` method.

```java
// This snippet display the currency as per the locale
import java.text.NumberFormat;
import java.util.Currency;

public class CurrencyClass{
 public static void main ( String args []){
 NumberFormat formatter = NumberFormat.getInstance () ;
   System.out.println ( formatter.getCurrency ()) ;
    }
}
```

Formatting Numbers 3-5

`public static final NumberFormat getInstance()`

❑ This method returns the default number format for the current default locale.

❑ The code snippet shows the use of `getInstance()` method.

```
// this snippet display the currency as per the locale //
import java.text.NumberFormat;
import java.util.Currency;

public class CurrencyClass{
 public static void main ( String args[]){
 NumberFormat formatter = NumberFormat.getInstance();
 System.out.println ( formatter.getCurrency());
 }
}
```

Formatting Numbers 4-5

```
public Number parse(String str) throws ParseException
```

❑ This method parses text from the beginning of the specified string to produce a number.

❑ The code snippet shows the use of `parse()` method.

```
NumberFormat nf = NumberFormat.getInstance ( );
Number myDemoNumber = nf.parse(myDemoString);
```

```
public void setCurrency(Currency currency)
```

❑ This method sets the currency used by the number format when formatting currency values.

❑ This does not update the minimum or maximum number of fraction digits used by the number format.

❑ The code snippet shows the use of `setCurrency()` method.

```
// Sets the currency to new amountCurrency
NumberFormat formatter =
NumberFormat.getInstance();

formatter.setCurrency(amountCurrency);
```

Formatting Percentages 1-2

❖ The format for displaying percentages can be changed using `getPercentInstance()` method of `NumberFormat` class.

❖ For example, with this format, a fraction as `0.82` can be displayed as `82%`.

❖ The two methods for formatting percentage are as follows:

`getPercentInstance()`	`getPercentInstance(Locale inLocale)`
• This method returns a percentage format for the current default locale. • **Syntax**: `public static final NumberFormat getPercentInstance()`	• This method returns a percentage format for the specified locale. • **Syntax**: `public static NumberFormat getPercentInstance(Locale inLocale)`

Formatting Percentages 2-2

❖ Figure depicts the code for formatting of percentages.

```
static public void displayPercent(Locale currentLocale) {

    Double percent = new Double(0.75);
    NumberFormat percentFormatter;
    String percentOut;

    percentFormatter = NumberFormat.getPercentInstance(currentLocale);
    percentOut = percentFormatter.format(percent);
    System.out.println(percentOut + "    " + currentLocale.toString());
  }
}

Output:

75%    en_US
```

Formatting Messages 1-2

Formatting also helps provide messages in user's language.

In order to format a message, the `MessageFormat` object is used.

The array of objects using the format specifiers embedded in the pattern is formatted by `MessageFormat.format()` method.

It returns the result as a `StringBuffer`.

Formatting Messages 2-2

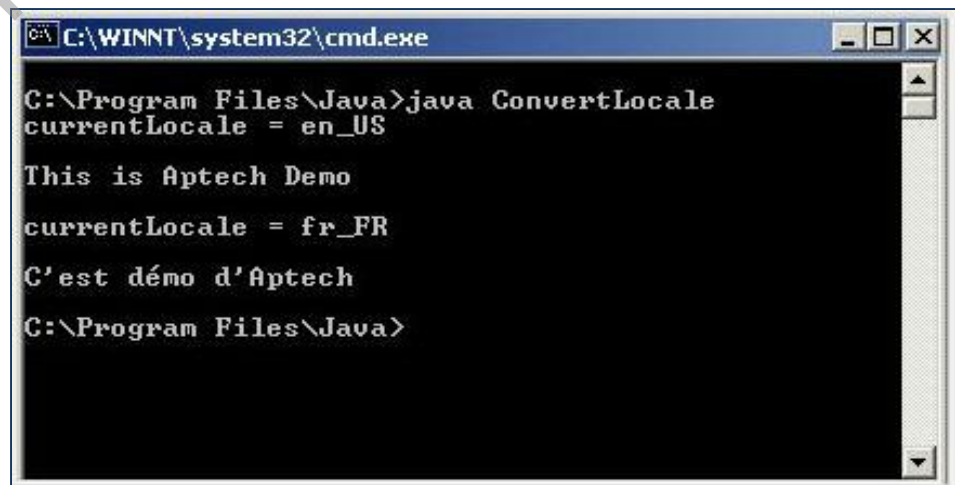❖ The classes for formatting the message are as follows:

MessageFormat

- This class provides a means to generate related messages in a language-neutral way.
- `MessageFormat` takes a set of objects, formats them, then inserts the formatted strings into the pattern at the appropriate places.
- It can also be used to construct messages displayed for end users.

MessageFormat.Field

- This class defines constants that are used as attribute keys in the `AttributedCharacterIterator` object returned from `MessageFormat.formatToCharacterIterator()`.

❖ Figure depicts message formatting.


```
C:\Program Files\Java>java ConvertLocale
currentLocale = en_US

This is Aptech Demo

currentLocale = fr_FR

C'est démo d'Aptech

C:\Program Files\Java>
```

Internationalizing JSP Pages

❖ JSP Standard Tag Library (JSTL) provides a set of internationalization or I18N tags that are used for applying various internationalization formats.

❖ The tag library helps reduce and manage the complexities of internationalized applications.

❖ Several tags available in I18n tag library are as follows:

- ❑ `formatDate`
- ❑ `formatNumber`
- ❑ `message`

Formatting Dates in JSP

❖ The tag `<fmt:formatDate>` is used for formatting dates and/or time in JSP for internationalization.

❖ The value attribute or the body content of the `<fmt:formatDate>` tag formats the date value.

❖ The formatted date is written to the JSP's writer.

❖ The value can also be stored in a string named `var` and an optional `scope` attribute.

❖ **Syntax**:

```
<fmt:formatDate value="date" [type="{time|date|both}"]

[dateStyle="{default|short|medium|long|full}"]

[timeStyle="{default|short|medium|long|full}"]

[pattern="customPattern"]

[timeZone="timeZone"]

[var="varName"]

[scope="{page|request|session|application}"]/>
```

❖ **Example:** `<fmt:formatDate value="${FinishDate}" type="date" dateStyle="full"/>`

Formatting Currencies in JSP 1-3

❖ The currencies can be formatted in JSP for internationalization using JSTL I18N tags.

❖ The `<fmt:setLocale>` stores the specified locale in the `javax.servlet.jsp.jstl.fmt.locale` configuration variable.

❖ The formatting is done as per the set locale.

❖ The tag `<fmt:formatNumber>` can be used to format the currencies.

❖ The number is specified to be formatted either with an EL expression in value attribute or as the tag's body content.

❖ The desired formatting is specified by the type attribute.

```
<fmt:setLocale>
```

❖ **Syntax**:

```
<fmt:setLocale value="locale"

[variant="variant"]

[scope="{page|request|session|application}"]/>
```

Formatting Currencies in JSP 2-3

`<fmt:formatNumber>`

❖ **Syntax**:

```
<fmt:formatNumber value="numericValue"

[type="{number|currency|percent}"]

[pattern="customPattern"]

[currencyCode="currencyCode"]

[currencySymbol="currencySymbol"]

[groupingUsed="{true|false}"]

[maxIntegerDigits="maxIntegerDigits"]

[minIntegerDigits="minIntegerDigits"]

[maxFractionDigits="maxFractionDigits"]

[minFractionDigits="minFractionDigits"]

[var="varName"]

[scope="{page|request|session|application}"]/>
```

❖ The code snippet demonstrates the formatting of currency in JSP.

```
//formatting for currency

<fmt:setLocale value="en_GB"/>

Formatting salary with Locale <B>en_GB</B> becomes :

<fmt:formatNumber type="currency" value="${salary}"
/><BR>
```

Formatting Percentages in JSP

❖ The `<fmt:formatNumber>` tag formats a number in integer, decimal, currency, and percentage.

❖ By specifying the type attribute in `<fmt:formatNumber>` percentage of a number can be obtained.

❖ This happens when the value is multiplied with hundred.

Example: `<fmt:formatNumber value="0.82" type="percent"/>`

❖ The `<fmt:message>` tag retrieves a message from a resource bundle and optionally, uses the `java.util.MessageFormat` class to format the message.

❖ The key attribute specifies the message key.

❖ If the `<fmt:message>` tag occurs within a `<fmt:bundle>` tag, the key is appended to the bundle's prefix, if there is one.

❖ If the `<fmt:message>` tag occurs outside of a `<fmt:setbundle>` tag, the bundle attribute must be present and must be an expression that evaluates to a `LocalizationContext` object.

❖ A variable is initialized with the `<fmt:setBundle>` tag.

❖ **Syntax**:

```
<fmt:message key="messageKey"

[bundle="resourceBundle"][var="varName"]

[scope="{page|request|session|application}"]/>
```

❖ The code snippet demonstrates the formatting of messages in JSP.

```
// This snippet formats the message
<fmt:message key="welcome">
  <fmt:param value="${userNameString}"/>
</fmt:message>
```

❖ Figure depicts formatting of messages.

Summary

❖ Internationalization can be defined as the method of designing an application that can be adapted to a region or a language without much change in the technology.

❖ Localization is a process of making a product or service, language, culture, and local 'look-and-feel' specific.

❖ Internationalization of server-side code reduces the task of writing multiple versions of a class for different locales.

❖ Resource bundles are used to achieve the locale specific output.

❖ Internationalization requires formatting of dates, numbers, currencies, and messages with the help of resource bundle.

❖ The internationalization or I18N tags in JSTL help to reduce and manage the complexities of internationalized applications. There are several tags available in I18N.