Distributed Programming in Java

Session: 7

Internationalization

Objectives

- Explain the need of Internationalization
- Explain the concept of Localization
- Describe how to create a properties file to store locale-specific data
- Describe how to define a Locale for a country
- Explain how to use the ResourceBundle class
- Explain how to retrieve locale-specific data from the properties file
- Explain the need and procedure for formatting numbers, currencies, and percentages
- Explain the need and procedure for formatting Date and Time
- Explain the need and procedure for using the MessageFormat

Need of Internationalization

- With the advent of the Internet, globalization of software products has become an imminent requirement.
- The main problems faced in globalization of software products are as follows:

1. Not all countries across the world speak or understand English language.
2. Symbols for currency vary across countries.
3. Date and Time are represented differently in some countries.
4. Spelling also varies amongst some countries.

Alternatives to Solve the Problem

- Two possible options for solving the problems faced are as follows:
- **Develop the entire product in the desired language:**

  This option will mean repetition of coding work. It is a time-consuming process and not an acceptable solution. Development cost will be much higher than the one time cost.

- **Translate the entire product in the desired language:**

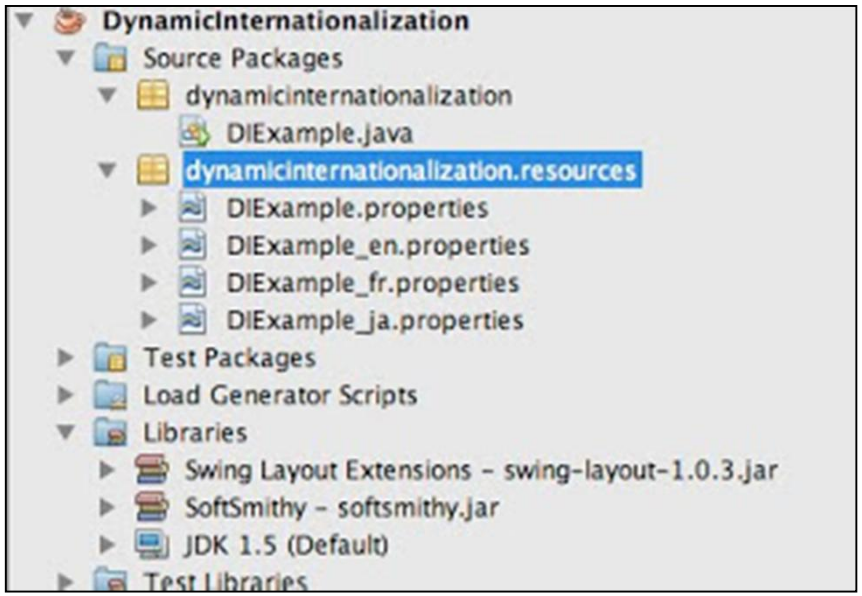
  Successful translation of the source files is very difficult. The menus, labels, and messages of most GUI components are hard-coded in the source code. It is not likely that a developer or a translator will have linguistic as well as coding skills.

- **Use internationalization:**

  Provide ability in application to adapt to various countries, languages.

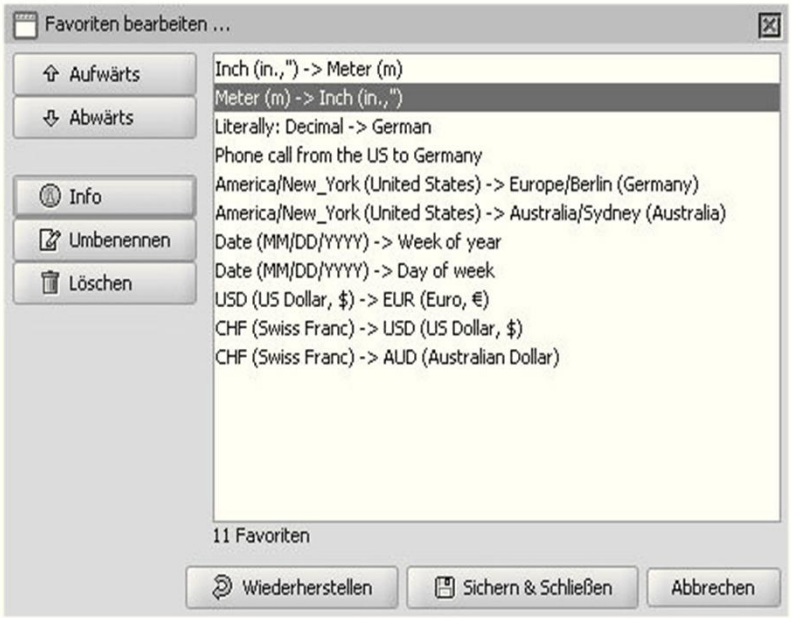
Meaning of Internationalization

- The process of creating applications which can adapt and adjust to various countries, languages, and regions is known as internationalization.
- Since there are 18 letters between the first 'I' and the last 'n', the word "internationalization" is often abbreviated as i18n.
- Internationalized software is developed independent of the countries or language of its users, and then localized for multiple countries or regions. Internationalized software should be developed such that it can be adapted without any engineering changes.



Meaning of Localization

- The process of adapting an application for a specific language or country by addition of locale-specific components and translation of text is referred to as localization and the word localization is often abbreviated as l10n.
- The translation of text is the most time-consuming part of the localization process.
- `Localization` also needs to see that the formatting of dates and currencies conforms to the requirements of the specific country.
- Figure displays an example of localization.



Benefits of I18N and l10N

- An internationalized and localized program has the following benefits:

1. With the addition of localized data, the same executable application can run worldwide.
2. Textual elements such as labels and messages for GUI components are not hard-coded in the program.
3. Textual elements are stored outside the source code and retrieved dynamically.
4. Support for new languages does not require recompilation.
5. Culturally dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
6. Internationalized programs are localized quickly.

Unicode Class

- `Unicode` provides a unique number for every character irrespective of platform, program, or language.
- Before `Unicode` was invented, there were several different encoding systems for assigning these numbers.
- No single encoding system contained enough number of characters to address international languages.
- These various encoding systems conflict with each other, that is, two encoding systems can have the same number for two different characters, or can use different numbers for the same character.
- `Unicode` is a 16-bit character encoding system that supports the world's major languages.
- The primitive data type `char` in Java is based on Unicode encoding.

Significance of Unicode

- The Java platform uses `Unicode` as its native character encoding.
- However, Java programs still need to handle characters in other encoding systems.
- The `String` class can be used to convert standard encoding systems to and from the `Unicode` system.
- To indicate `Unicode` characters that cannot be represented in ASCII, such as ö, you use the \uXXXX escape sequence.
- Each X in the escape sequence is a hexadecimal digit.
- The following code snippet shows how to indicate the ö character with an escape sequence.
- Code Snippet shows how to indicate the ö character with an escape sequence.

Code Snippet

```
String str = "\u00F6";
char c = '\u00F6';
Character letter = new Character('\u00F6');
```

Introduction to Properties File

- A properties file is a plain text file, which contains the information that is ultimately translated into an Internationalized program.
- A default properties file with any name, but having extension as .properties contains the '`key = value`' pair in English.
- The keys on the left-hand side are not translated, whereas the values on the right hand side are translated.
- These translated values are kept in different files for different languages and countries. The naming convention for these files is `DefaultFileName_lc_CC.properties`.
- The '`lc`' represents the language code in lower case and '`CC`', the country code in upper case.

Creating the Properties File

- To create a properties file, you can use any plain text editor such as Notepad.
- The default properties file contains the '`key = value`' pair in English.
- For each language and country, you create a properties file with the naming convention.
- A country can have more than one language.
- Some examples of properties files which contain messages are as follows:
  - `MessageBundle.properties`

```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

  - `MessageBundle_fr_FR.properties`

```
greetings = Bonjour.
farewell = Au revoir.
inquiry = Comment allez-vous?
```

Defining the Locale

- A `Locale` is simply an identifier for a particular combination of language and region.
- A `java.util.Locale` class object represents a specific geographical, political, or cultural region.
- Any operation that requires a locale to perform its task is said to be locale-sensitive.
- These operations use the `Locale` object to tailor information for the user.
- For example, displaying a number is a locale-sensitive operation.
- The number should be formatted according to the customs and conventions of a user's native country, region, or culture.
- A `Locale` object is created using the constructors:
  - `public Locale(String language, String country)`
  - `public Locale(String language)`

Methods of Class Locale

- `public static Locale getDefault()`: Gets the default Locale for this instance of the Java Virtual Machine.
  - Example: `Locale currentLocale = Locale.getDefault();`
- `public final String getDisplayCountry()`: Returns the name of the country for the current Locale, which is appropriate for display to the user. For example, if the default locale is `fr_FR`, this method will return France.
  - Example:

```
Locale currentLocale = Locale.getDefault();
String strCountry = currentLocale.getDisplayCountry();
```

- `public final String getDisplayLanguage()`: Returns the name of the language for the current Locale, which is appropriate for display to the user. For example, if the default locale is `fr_FR`, this method will return French.
  - Example:

```
Locale currentLocale = Locale.getDefault();
String strLanguage = currentLocale.getDisplayLanguage();
```

Locale Class Methods

- The `Locale` class has several methods that can be used for localization of an internationalized application.
- The methods of the `Locale` class which will be discussed in the session are:
  - `getAvailableLocales()`
  - `getCountry()`
  - `getLanguage()`
  - `getDisplayName()`
  - `setDefault()`

Language and Country Codes

- The international standard defines the language and region codes.
- This standard assigns two- and three-letter codes to languages of the world.
- Table lists a few of the examples for language codes.

Language	Code
English	En
Arabic	Ar
German	De

- Similarly, two-and three-letter abbreviations are used for each country or major region of the world.
- Table lists a few of the examples for country codes.

Country	Code
United States	US
Canada	CA
France	FR
Japan	JP

- The elements such as date, time, currency, and numbers are country specific and dependent on the country codes.

Enabled Script Support

- A script is the writing form for a language.
- In GUI applications, text components need to support multiple locales.
- **AWT Components**
  - AWT Components can render scripts supported by the locale host.
  - Example: If the host system is localized to Arabic, then AWT will display Arabic text components.
  - The same AWT component cannot display the text in a script different from the localized host.
- **JFC Components**
  - JFC Swing component supports multiple scripts due to their independence from the host platform and their use of the Unicode Charset.
- Table lists some of the scripts supported for text display.

Writing System	Language Support
Arabic	Arabic
Chinese (simplified)	Chinese
Chinese (traditional)	Chinese
Hebrew	Hebrew
Japanese	Japanese
Korean	Korean

Creating the ResourceBundle

- The `ResourceBundle` class is used to retrieve locale-specific information from the properties file.
- This information allows you to write applications that can be:

1. Localized or translated into different languages.
2. Handled for multiple locales at the same time.
3. Supported for more locales later.

Using the ResourceBundle Class

- The `ResourceBundle` class has a static and final method called `getBundle()` which helps us retrieve a `ResourceBundle` instance.

  Syntax:

  ```
  public static final ResourceBundle getBundle(String baseName, Locale locale)
  ```

- Code Snippet shows how to retrieve the locale-specific information from `MessageBundle_fr_FR.properties` file.

Code Snippet

```
String language = "fr";
String country = "FR";
Locale currentLocale;
ResourceBundle messages;
// Creates a Locale object with language French and country France
   currentLocale = new Locale(language, country);
// Retrieves the locale-specific information
   messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
```

Fetching the Text from ResourceBundle [1-2]

- To retrieve the locale-specific data from the properties file, you first create a `ResourceBundle` class object.
- Once you have a `ResourceBundle` object, you invoke the following methods:
  - `public final String getString(String key)`: The `getString()` method takes a string as an argument, this argument specifies the key from the properties file whose value is to be retrieved.
  - Code Snippet shows how to retrieve individual locale-specific data.

Code Snippet

```
// Create a Locale object for the French language and France country
   Locale currentLocale = new Locale("fr", "FR");
// Create a ResourceBundle object from a MessageBundle.properties file
    ResourceBundle messages =
                ResourceBundle.getBundle("MessageBundle",currentLocale);
// Retrieve individual locale-specific data
   String greeting = messages.getString("greetings");
   String inquiry = messages.getString("inquiry");
   String farewell = messages.getString("farewell");
```

Fetching the Text from ResourceBundle [2-2]

- `public abstract Enumeration<String> getKeys()`: The `getKeys()` method returns an enumeration object representing all the available keys in the properties file.
- Code Snippet shows how to iterate through all the keys and retrieve their values.

Code Snippet

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

Methods of ResourceBundle Class [1-3]

- The `ResourceBundle` class has methods which can be used for localization of an internationalized application.
- The methods of the `ResourceBundle` class which will be discussed in the session are:
  - `public Locale getLocale()`: The `getLocale()` method is used to retrieve the locale object of the `ResourceBundle` instance.
  - Code Snippet shows how to retrieve the locale object of resource bundle.

Code Snippet

```
String language = "fr";
String country = "FR";
Locale currentLocale;
ResourceBundle messages;
// Creates a Locale object with the language French and country
   France currentLocale = new Locale(language, country);
. . . . . .
// Retrieves the resource bundle for the default locale
   messages = ResourceBundle.getBundle("MessagesBundle");
// Retrieve the locale of the resource bundle
   Locale locale = messages.getLocale();
```

Methods of ResourceBundle Class [2-3]

```
// Check if the default locale corresponds to the French
//locale.
if (currentLocale.getCountry().equals(locale.getCountry) &&
currentLocale.getLanguage().equals(locale.getLanguage) ) {
        // The default locale corresponds to French locale
        . . .
}
. . .
```

- `public final String[] getStringArray(String key)` : The static method `getStringArray()` is used to retrieve a string array for the specified key from the resource bundle or one of its parents.
- Code Snippet shows retrieves the resource bundle.

Code Snippet

```
// Retrieves the resource bundle for the default locale
ResourceBundle messages =
ResourceBundle.getBundle("MessagesBundle");
String[] values = messages.getStringArray("greetings");
```

Methods of ResourceBundle Class [3-3]

- `protected void setParent(ReourceBundle parent)` : The static method `setParent()` is used to set the parent bundle of a resource bundle object. The parent bundle is searched for a particular resource if the resource is not found in the current resource bundle.
- Code Snippet shows retrieves the resource bundle for the messages.

Code Snippet

```
ResourceBundle messages;
ResourceBundle warnings;
// Retrieves the resource bundle for the messages
    messages = ResourceBundle.getBundle("MessagesBundle");
// Retrieves the resource bundle for the warnings
    warnings = ResourceBundle.getBundle("WarningsBundle");
// Set the warnings bundle as the parent
    messages.setParent(warnings);
```

Internationalization Elements

- Normally, the captions of a GUI component contain text which can be easily localized. However, these captions may also contain elements such as date, numerals, percentage factor, and measurement and so on. Since, such elements may vary with culture, region, and language, it is required to format the captions of the GUI components.
- Formatting the captions of the GUI components ensures that the look and feel of the application is in a locale-sensitive manner.
- The code, which handles the GUI, is locale-independent, without the need to write formatting routines for specific locales.

Need for Formatting

- The format of numbers, currencies, and percentages vary with culture, region, and language.
- Hence, it is necessary to format them before they are displayed.
- For example, the number 12345678 should be formatted and displayed as 12,345,678 in the US and 12.345.678 in Germany.
- Similarly, the currency symbols and methods of displaying the percentage factor also vary with region and language.
- Formatting is required to make an internationalized application independent of local conventions with regards to decimal-point, thousands-separators, and percentage representation.

```
123á456    fr_FR
345á987,246    fr_FR

123.456    de_DE
345.987,246    de_DE

123,456    en_US
345,987.246    en_US
```

Need for formatting

Numbers, Currencies, and Percentages

The `NumberFormat` class is used to create locale-specific formats for the following:

- **Numbers:** The `NumberFormat` class has a static method `getNumberInstance()`. The `getNumberInstance()` returns an instance of a `NumberFormat` class initialized to default or specified locale.
- **Currencies:** The `NumberFormat` class has a static method `getCurrencyInstance()` which takes an instance of `Locale` class as an argument. The `getCurrencyInstance()` returns an instance of a `NumberFormat` class initialized to the specified locale.
- **Percentages**: This class has a static method `getPercentInstance()` which takes an instance of `Locale` class as an argument. The `getPercentInstance()` returns an instance of a `NumberFormat` class initialized to the specified locale.

Date and Time

- The date and time format should conform to the conventions of the end user's locale.
- The date and time format varies with culture, region, and language.
- Hence, it is necessary to format them before they are displayed.
- For example, in Germany, the date can be represented as 20.04.07, whereas in US, it is represented as 04/20/07.
- Java provides the `java.text.DateFormat` and `java.text.SimpleDateFormat` class to format date and time.

Formatting Date and Time

- The `DateFormat` class is used to create locale-specific formats for date.
- You then invoke the `format()` method of the `NumberFormat` class.
- The number to be formatted is passed as an argument.
- The argument can be a primitive data type or a wrapper class object.
- The method `getDateInstance()` returns an instance of the class `DateFormat` for the specified style and locale.
- Code Snippet shows how to retrieve a `DateFormat` object.

Code Snippet

```
Date today;
String strDate;
DateFormat dateFormatter;
Locale locale = new Locale("fr", "FR");
dateFormatter = DateFormat.getDateInstance(DateFormat.MEDIUM,
locale); t
oday = new Date();
strDate = dateFormatter.format(today);
System.out.println(strDate);
```

Benefits and Formatting Date and Time

- The `DateFormat` class has methods which can be used to format the date and time for a specific locale.
- **`getDateTimeInstance()`**
  - The static method `getDateTimeInstance()` is used to retrieve a date/time formatter with the default formatting style for the default locale.
  - Code Snippet shows how to retrieve a date and time formatter with default style for default locale.

Code Snippet

```
Date today;
String strDate;
DateFormat dateFormatter;
Locale locale = new Locale("fr", "FR");
dateFormatter = DateFormat.getDateTimeInstance();
today = new Date();
strDate = dateFormatter.format(today);
System.out.println(strDate);
```

DateFormat Class and its Methods [1-2]

- The `SimpleDateFormat` class is a concrete implementation of the abstract class `DateFormat`.
- The `SimpleDateFormat` class has methods to parse and convert a text to date.
- The `parse()` method parses text from a string to produce a `Date` object.
- This method parses the text from a given position specified by an object of the `ParsePosition` class.
- If parsing succeeds, the index of the parse position is updated to the last character in the text where the date ends.
- Code Snippet shows how to parse a text and return a date.

Code Snippet

```
Date today;
String text;
SimpleDateFormat dateFormat;
ParsePosition pos;
try  {
        // Assign text containing a valid date
```

DateFormat Class and its Methods [2-2]

```
    text = "Today the date is 26/06/2007 and tomorrow…";

 // Create a SimpleDateFormat object with the pattern dd/mm/yy
    dateFormat = new SimpleDateFormat("dd/mm/yyyy");
// Specify the position of the date in text,18 in the above text
    pos = new ParsePosition(18);
// Parse the text to return a date object
    today = dateFormat.parse(text, pos);
    System.out.println("Date : " + today);
}
catch (NullPointerException ex)  {
 System.out.println("Exception : could not parse the date.");
}
```

- Output:

```
<terminated> snippet40 [Java Application] C:\Program Files\Java
Date : Fri Jan 26 00:06:00 IST 2007
```

Formatting Messages

- Displaying messages such as status and error messages are an integral part of any software.
- If the messages are predefined such as “Your License has expired”, they can be easily translated into various languages.
- However, if the messages contain variable data, it is difficult to create grammatically correct translations for all languages.
- The position of verbs and the variable data varies in different languages.

On 06/03/2007 we detected 10 viruses

Sur 06/03/2007 nous avons détecté le virus 10

Auf 06/03/2007 ermittelten wir Virus 10

Formatting messages

Need for MessageFormat Class

- It is not always possible to create a grammatically correct sentence with concatenation of phrases and variables.
- The approach of concatenation works fine in English, but it will not work for languages in which the verb appears at the end of the sentence.
- If the word order in a message is hard-coded, you will not be able to create grammatically correct translations for all languages.
- The solution is to use the `MessageFormat` class to create a compound message.
- Code Snippet shows how to concatenate several elements of a message.

Code Snippet

```
int num = 10;
Date date = new Date();
String text = messageBundle.getString("On")+date.toString()+
messageBundle.getString("we detected") + numVirus +
messageBundle.getString("virus");
```

MessageFormat Class

- To use the `MessageFormat` class, you typically perform the following steps:

1. Identify the variables in the message.
2. Create a template.
3. Create an `Object` array for variable arguments.
4. Create a `MessageFormat` instance.
5. Apply and `Format` the pattern.

Procedure to Use MessageFormat Class [1-2]

- **Step 1**: The first step involves creating a template for the following text:

  At 6:41 PM on April 25, 2007, we detected 7 virus on the disk D:

  The template for this message is as follows:

```
template = At {2,time,short} on {2,date,long}, we detected
        {1,number,integer} virus on the disk {0}
```

- **Step 2**: The second step involves creating an array of objects containing variable arguments. The following snippet creates an `Object` array for the template:

```
template = At {2,time,short} on {2,date,long}, we detected
{1,number,integer} virus on the disk {0}
Object[] messageArguments = {  messages.getString("disk"),
new
Integer(7),new Date() };
```

Procedure to Use MessageFormat Class [2-2]

- **Step 3**: The third step is to create an instance of `MessageFormat` class. The following snippet creates an object of `MessageFormat` class:

```
MessageFormat formatter = new MessageFormat("");
```

- **Step 4**: The last step is to format the text using `MessageFormat` instance.

  The following snippet shows the complete code required to successfully translate the following message:

  At 10:53 AM on April 26, 2007, we detected 7 virus on the disk D:

```
Object[] messageArguments = {  messages.getString("disk"),
new Integer(7),  new Date();
ResourceBundle messages =
ResourceBundle.getBundle("MessageFormatBundle", currentLocale);
MessageFormat formatter = new MessageFormat("");
formatter.setLocale(currentLocale);
formatter.applyPattern(messages.getString("template"));
String output = formatter.format(messageArguments);
System.out.println(output);
```

ParsePosition Class

- The `ParsePosition` class is used to specify the initial position from where to start parsing.
- The `ParsePosition` has one constructor which takes an argument specifying the initial position to start parsing.

Syntax:

```
public ParsePosition(int initialPosition)
```

- Code Snippet shows how to specify the initial position for parsing the text.

Code Snippet

```
ParsePosition pos;
. . .
// Start the parsing at zero position
pos = new ParsePosition(0);
```

Summary

- Internationalization is a way of designing an application, which adapts to various countries, languages, and regions. Such software is developed independent of the countries or language of its users, and then localized for multiple languages.
- Internationalization process involves creating a properties file to store locale-specific data and defining a `Locale` for a country.
- We then create a `ResourceBundle` and use it to retrieve locale-specific information from the properties file.
- The GUI component captions contain text, which can be easily localized.
- However, these captions may also contain elements such as date, numerals, percentage factor, measurements, and so on.
- Since these elements may vary with culture, region, and language, it is required to format the captions of GUI components.