Globalization in java

Globalization in java

- An application that can present information to users according to regional cultural conventions is said to be *globalized*.
- In a globalized application, a user in one region sees error messages, output, and interface elements in the requested language.
- Globalization consists of two phases: *internationalization* and *localization*

Internationalization in java (i18n)

- Internationalization is the process of designing an application for multicultural support, so that it can be adapted to various languages and regions without engineering changes.
- Some of the characteristics of Internationalized program are: Textual elements, such as status messages are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically., Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.It can be localized quickly.

Localization in java (l10n)

- *Localization* is the process of adapting software for a specific region or language by adding locale-specific components and translating text.
- The primary task of localization is translating the user interface elements and documentation.
- The better internationalized an application is, the easier it is to localize it for a particular language and character encoding scheme.

Before Internationalization

```
public class NotI18N {

    static public void main(String[] args) {

        System.out.println("Hello.");

        System.out.println("How are you?");

        System.out.println("Goodbye.");

    }

}
```

Before Internationalization

- On the last slide, there is a program written that displays three messages, such as hello, how are u and goodbye.
- Now what if this program needs to display these same messages for people living in France and Germany that is in french and german respectively.
- Unfortunately the programming staff is not multilingual, so will need help translating the messages into French and German. but the translators aren't programmers.
- so we have to move the messages out of the source code and into text files that the translators can edit.So it make it happen, we need internationalization to make our application globally used.

For Internationalization

- Locale currentLocale=new Locale(language, country);
- ResourceBundle messages = ResourceBundle.getBundle("localeDemo/MessagesBundle", currentLocale);
- ResourceBundle is used to fetchdata from the properties file.
- Both of these classes belongs to java.util.* package.

Internationalization sample code

```
import java.util.*;
public class I18NSample {
    static public void main(String[] args) {

        String language;
        String country;

        if (args.length != 2) {
            language = new String("en");
            country = new String("US");
        } else {
            language = new String(args[0]);
            country = new String(args[1]);
        }
        Locale currentLocale;
        ResourceBundle messages;
        currentLocale = new Locale(language, country);
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
```

Internationalization

- This is the source code for the internationalized program. Notice that the text of the messages is not included in the software code.
- Now to translate the messages into french and german, we should create properties files, which are the text files the translators can edit.
- The internationalized program is flexible; it allows the end user to specify a language and a country on the command line.
- To compile and run this program, you need these source files:
  - `I18NSample.java`
  - `MessagesBundle.properties`
  - `MessagesBundle_de_DE.properties`
  - `MessagesBundle_fr_FR.properties`

Formatting numbers and currencies according to Locale

- NumberFormat is the abstract base class for all number formats. This class provides the interface for formatting and parsing numbers.
- Use getInstance or getNumberInstance to get the normal number format. Use getIntegerInstance to get an integer number format. Use getCurrencyInstance to get the currency number format.
- By invoking the methods provided by the NumberFormat class, you can format numbers, currencies, and percentages according to Locale.
- You can use the NumberFormat methods to format primitive-type numbers, such as double, and their corresponding wrapper objects, such as Double.

```java
import java.util.*;

public class NumberFormatDemo {

  static public void displayNumber(Locale currentLocale) {
    Integer a = new Integer(123456);
    Double b = new Double(345987.246);

    NumberFormat numberFormatter = NumberFormat.getNumberInstance(currentLocale);

    System.out.println(numberFormatter.format(a) + "   " + currentLocale.toString());
    System.out.println(numberFormatter.format(b) + "   " + currentLocale.toString());
  }

  static public void displayCurrency(Locale currentLocale) {
    Double currencyAmount = new Double(9876543.21);

    Currency currentCurrency = Currency.getInstance(currentLocale);
    NumberFormat currencyFormatter = NumberFormat.getCurrencyInstance(currentLocale);

    System.out.println(currentLocale.getDisplayName() + ", "
        + currentCurrency.getDisplayName() + ": "
        + currencyFormatter.format(currencyAmount));
  }
```

```java
static public void main(String[] args) {
  ArrayList<Locale> locales = new ArrayList<>();
  locales.add(new Locale("fr","FR"));
  locales.add(new Locale("de","DE"));
  locales.add(new Locale("en","US"));

  for (int i = 0; i < locales.size(); i++) {
    displayNumber(locales.get(i));
  }
  for (int i = 0; i < locales.size(); i++) {
    displayCurrency(locales.get(i));
  }
}
```

OUTPUT:

123 456 fr_FR
345 987,246 fr_FR
123.456 de_DE
345.987,246 de_DE
123,456 en_US
345,987.246 en_US
French (France), Euro: 9 876 543,21 €
German (Germany), Euro: 9.876.543,21 €
English (United States), US Dollar: $9,876,543.21

Formatting numbers and currencies according to Locale

- The previous code example formats a Double according to Locale. Invoking the getNumberInstance method returns a locale-specific instance of NumberFormat. The format method accepts the Double as an argument and returns the formatted number in a String.
- You format currencies in the same manner as numbers, except that you call getCurrencyInstance to create a formatter. When you invoke the format method, it returns a String that includes the formatted number and the appropriate currency sign.

Formatting the numbers in the user defined pattern

```
package localeDemo;

import java.util.*;

public class DecimalFormatDemo {

   static public void customFormat(String pattern, double value ) {
      DecimalFormat myFormatter = new DecimalFormat(pattern);
      System.out.println(value + "  " + pattern + "  " +
                                    myFormatter.format(value));
   }

   static public void main(String[] args) {

      customFormat("###,###.###", 123456.789);
      customFormat("###.##", 123456.789);
      customFormat("000000.000", 123.78);
      customFormat("$###,###.###", 12345.67);
      customFormat("\u00a5###,###.###", 12345.67);
   }
}
```

Output:

123456.789 ###,###.### 123,456.789

123456.789 ###.## 123456.79

123.78 000000.000 000123.780

12345.67 $###,###.### $12,345.67

12345.67 ¥###,###.### ¥12,345.67

Formatting date and time according to the locale

```
static public void showBothStyles(Locale currentLocale) {
   Date today=new Date();
   DateFormat formatter;
   int[] styles = {
      DateFormat.DEFAULT,
      DateFormat.SHORT,
      DateFormat.MEDIUM,
      DateFormat.LONG,
      DateFormat.FULL
   };
   System.out.println();
   System.out.println("Locale: " + currentLocale.toString());
   System.out.println();
   for (int k = 0; k < styles.length; k++) {
      formatter = DateFormat.getDateTimeInstance(styles[k], styles[k], currentLocale);
      System.out.println(formatter.format(today));
   }

   static public void main(String[] args) {
      Locale[] locales = {new Locale("fr","FR"),new Locale("de","DE"), new Locale("en","US")};
      showBothStyles(new Locale("en","US"));
      showBothStyles(new Locale("fr","FR"));
   }
}
```

Output

```
Locale: en_US

Aug 24, 2022 5:00:24 PM
8/24/22 5:00 PM
Aug 24, 2022 5:00:24 PM
August 24, 2022 5:00:24 PM IST
Wednesday, August 24, 2022 5:00:24 PM IST

Locale: fr_FR

24 août 2022 17:00:24
24/08/22 17:00
24 août 2022 17:00:24
24 août 2022 17:00:24 IST
mercredi 24 août 2022 17 h 00 IST
```

Formatting date and time according to the locale

- As shown in the last slide, to print all the styles of both date and time, “DateFormat.getDateTimeInstance(styles[k], styles[k], currentLocale);” is used.
- To get only the instance of date in all the styles, we need to use “DateFormat.getDateInstance(styles[k], currentLocale); ”.
- Similarly, to get only the instance of time in all the styles, we need to use “DateFormat.getTimeInstance(styles[k], currentLocale);”.

To completely internationalize an existing program, take the following steps:

- Identify Culturally Dependent Data : Messages, Date, Time, Currency, Measurements etc
- Isolate Translatable Text in Resource Bundles
- Deal with Compound Messages
- Format Numbers and Currencies
- Format Dates and Times
- Use Unicode Character Properties
- Compare Strings Properly
- Convert Non-Unicode Text

Thank you