Internationalization and Localization

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Introduction

Internationalization (i18n) is the process of designing an application so that it can be adapted to different languages and regions, without requiring engineering changes.

Localization (l10n) is the process of adapting software for a specific region or language by adding locale-specific components and translating text.

Organization of Presentation

- What is i18n?
- Java example of messages
- What is a “locale”?
- Formatting data in messages
- Translation issues
- Date/Time/Currency/etc
- Unicode and support in Java
- Iteration through text

Why is i18n important?

- Build once, sell anywhere…
- Modularity demands it!
  - Ease of translation
- “With the addition of localization data, the same executable can be run worldwide.”

Characteristics of i18n...

- Textual elements such as status messages and the GUI component labels are not hardcoded in the program. Instead, they are stored outside the source code and retrieved dynamically.

- Support for new languages does not require recompilation.

- Other culturally-dependent data, such as dates and currencies, appear in formats that conform to the end-user's region and language.

Java Class : Locale

Locale : A Locale object represents a specific geographical, political, or cultural region.

The Locale class provides three constructors:

Locale(String language)

Locale(String language, String country)

Locale(String language, String country, String variant)

Java Classes: ResourceBundle

ResoureBundle: Resource bundles contain locale-specific objects

- **public static ResourceBundle getBundle(String basename)** returns the instance of the ResourceBundle class for the default locale
- **public static ResourceBundle getBundle(String basename, Locale locale)** returns the instance of the ResourceBundle class for the specified locale
- **public String getString(String key)** returns the value for the corresponding key from this resource bundle

The name of the properties file should be **filename_languagecode_country** code

Java Example: Messages...

Before:

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

Too much code!

After:

```
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"));
   }
```

Sample Run…

```
% java I18NSample fr FR
Bonjour.
Comment allez-vous?
Au revoir.

% java I18NSample en US
Hello.
How are you?
Goodbye.
```

1. So What Just Happened?

- Created `MessagesBundle_fr_FR.properties`, which contains these lines:

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

(What the translator deals with.)

- In the English one?

2. Define the locale...

- Look!

```
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"));
   }
```

3. Create a ResourceBundle...

- Look!

```
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"));
   }
```

4. Get the Text from the ResourceBundle...

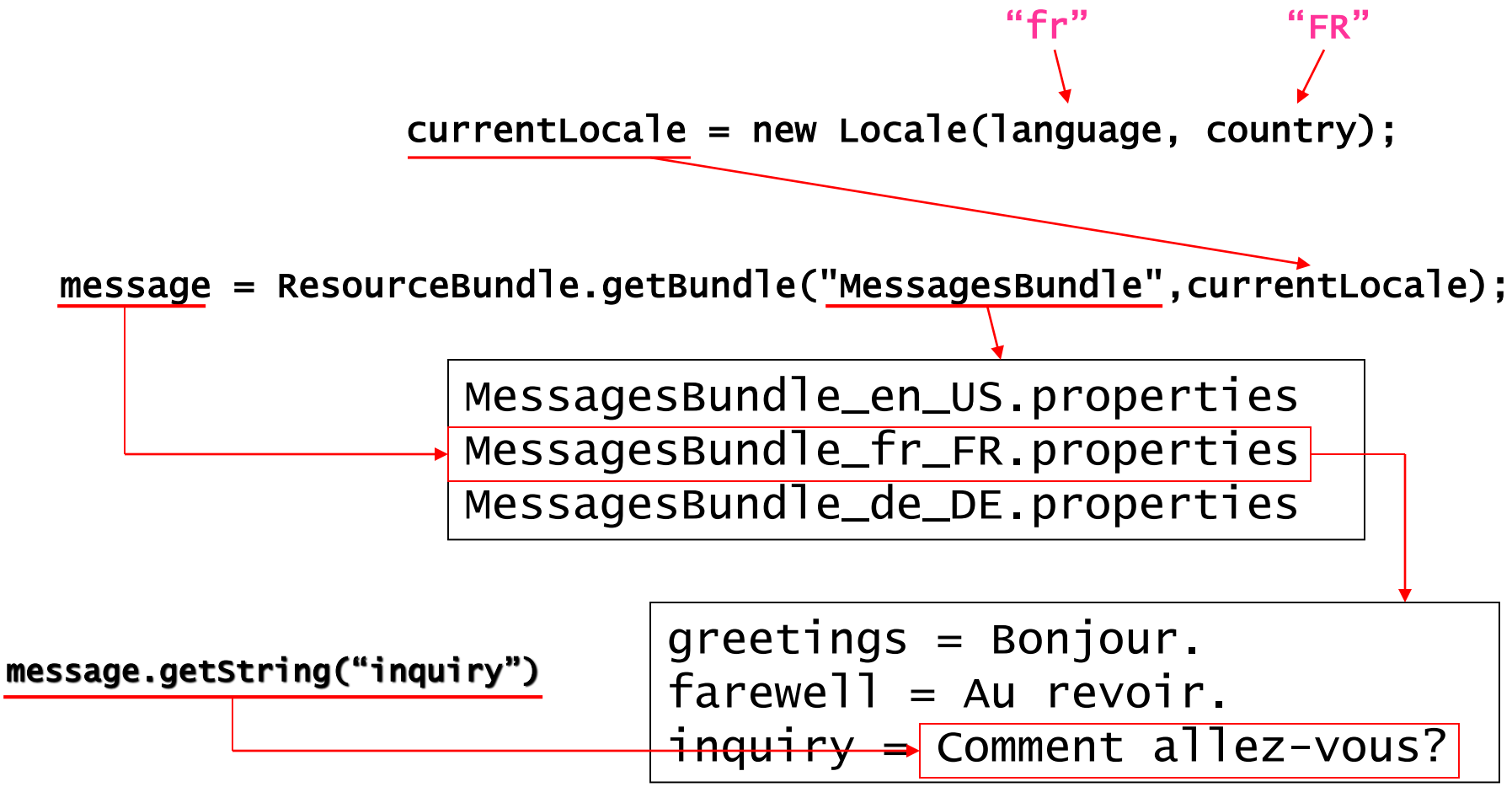
- Look!

```
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"));
   }
```

What is a “locale”?

- Locale objects are only identifiers.
- After defining a Locale, you pass it to other objects that perform useful tasks, such as formatting dates and numbers.
- These objects are called locale-sensitive, because their behavior varies according to Locale.
- A ResourceBundle is an example of a locale-sensitive object.

Did you get that?



Got a program…
need to…

- What do I have to change?
- What's easily translatable?
- What's NOT?
  - "It said 5:00pm on that $5.00 watch on May 5th!"
  - "There are 5 watches."
- Unicode characters.
- Comparing strings.

What do I have to change?

- Just a few things…
  - messages
  - labels on GUI components
  - online help
  - sounds
  - colors
  - graphics
  - icons
  - dates
  - times

more…

- numbers
- currencies
- measurements
- phone numbers
- honorifics and personal titles
- postal addresses
- page layouts

What's easily translatable?

Isolate it!

- Status messages
- Error messages
- Log file entries
- GUI component labels
  - **BAD!**

```
Button okButton = new Button("OK");
```

  - **GOOD!**

```
String okLabel = ButtonLabel.getString("OkKey");
Button okButton = new Button(okLabel);
```

Numbers and Currencies!

- What's wrong with my numbers?
  - We say:

```
345,987.246
```

  - Germans say:

```
345.987,246
```

  - French say:

```
345 987,246
```

Numbers... 123456...

- Supported through NumberFormat!

```
Locale[] locales = NumberFormat.getAvailableLocales();
```

create custom formats if needed.

```
Double amount = new Double(345987.246);
NumberFormat numberFormatter;
String amountOut;

numberFormatter = NumberFormat.getNumberInstance(currentLocale);
amountOut = numberFormatter.format(amount);
System.out.println(amountOut + "    " + currentLocale.toString());
```

```
345 987,246    fr_FR
345.987,246    de_DE
345,987.246    en_US
```

Money!

- Supported with:
  NumberFormat.getCurrencyInstance!

```
Double currency = new Double(9876543.21);
NumberFormat currencyFormatter;
String currencyOut;

currencyFormatter = NumberFormat.getCurrencyInstance(currentLocale);
currencyOut = currencyFormatter.format(currency);
System.out.println(currencyOut + "    " + currentLocale.toString());
```

```
9 876 543,21 F    fr_FR
9.876.543,21 DM    de_DE
$9,876,543.21    en_US
```

Percents? 100%...

- Supported with:
  NumberFormat.getPercentInstance!

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

percentFormatter = NumberFormat.getPercentInstance(currentLocale);
percentOut = percentFormatter.format(percent);
```

“A Date and Time…

- Supported with:
  - DateFormat.getDateInstance

```
DateFormat dateFormatter =
            DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
```

  - DateFormat.getTimeInstance

```
DateFormat timeFormatter =
            DateFormat.getTimeInstance(DateFormat.DEFAULT, currentLocale);
```

  - DateFormat.getDateTimeInstance

```
DateFormat dateTimeFormatter = DateFormat.getDateTimeInstance(
            DateFormat.LONG, DateFormat.LONG, currentLocale);
```

Date example…

- Supported with: DateFormat.getDateInstance!

```
Date today;
String dateOut;
DateFormat dateFormatter;

dateFormatter = DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
today = new Date();
dateOut = dateFormatter.format(today);

System.out.println(dateOut + "    " + currentLocale.toString());
```

```
9 avr 98    fr_FR
9.4.1998    de_DE
09-Apr-98    en_US
```


SISOFT
Skilling People...