**Chapter 12: Internationalization** 

- Processing Date and Time
  - -Locale
  - -Date
  - TimeZone
  - Calendar and GregorianCalendar
  - DateFormat and SimpleDateFormat
- Formatting Numbers
- Resource Bundles

# Java's International Support

#### 1. Use Unicode

2. Provide the Locale class to encapsulate information about a specific locale.

3. Use the ResourceBundle class to separate locale-specific information such as status messages and the GUI component labels from the program.

# The Locale Class

A Locale object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called *locale-sensitive*. You can use Locale to tailor information to the user.



# Creating a Locale

To create a Locale object, you can use the following constructor in Locale class:

Locale(String language, String country)

Locale(String language, String country, String variant)

Example:

new Locale("en", "US");

new Locale("fr", "CA");



# The Date Class

The Date class represents a specific instant in time, with millisecond precision. You can construct a Date object using one of the following two constructors in this class:

@ public Date();

@ public Date(long time);



# The TimeZone Class

The class TimeZone represents a time zone offset; it also figures out daylight-saving time. Typically, you can get a TimeZone object using its class method getDefault(), which creates a TimeZone object based on the time zone in which the program is running.



# Creating a TimeZone

You can also get a TimeZone object by using the class method getTimeZone(), along with a time zone ID. For example, the time zone ID for central standard time is CST. Therefore, you can get a CST TimeZone object with the following:

TimeZone tz = TimeZone.getTimeZone("CST");

# The Calendar Class

A Date object represents a specific instant in time with millisecond precision. Calendar is an abstract base class for converting between a Date object and a set of integer fields, such as year, month, day, hour, minute, and second.



## The GregorianCalendar Class

Subclasses of Calendar interpret a Date according to the rules of a specific calendar system. For example:

GregorianCalendar(TimeZone tz, Locale locale)

This constructs a GregorianCalendar object based on the current time in the given time zone with the given locale.

# The get () Method

public final int get(int field)

This retrieves the value for a given time field. The parameter field is a constant, such as YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, DAY\_OF\_WEEK, DAY\_OF\_MONTH, DAY\_OF\_YEAR, WEEK\_OF\_MONTH, WEEK OF YEAR, and so on.



#### GregorianCalendar Example

GregorianCalendar rightNow = new
GregorianCalendar();

System.out.println("week of the year is

"+rightNow.get(GregorianCalendar.WEEK\_OF\_YEAR));

This displays the week of the year for the current time.



## The DateFormat Class

The DateFormat class is an abstract class that provides many class methods for obtaining default date and time formatters based on the default or a given locale and a number of formatting styles, including FULL, LONG, MEDIUM, and SHORT.



#### DateFormat Formats

- SHORT is completely numeric, such as
   12.13.52 or 3:30pm
- MEDIUM is longer, such as Jan 12, 1952
- LONG is even longer, such as January 12,
   1952 or 3:30:32pm
- FULL is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST



## Creating a DateFormat

You can use the getDateTimeInstance() method to obtain a DateFormat object:

public static final DateFormat getDateTimeInstance
 (int dateStyle, int timeStyle, Locale aLocale)

This gets the date and time formatter with the given formatting styles for the given locale.

## The SimpleDateFormat Class

The date and time formatting subclass, such as SimpleDateFormat, enables you to choose any userdefined patterns for date and time formatting. To specify the time format, use a time pattern string:

formatter = new SimpleDateFormat("yyyy.MM.dd G
 'at' hh:mm:ss Z");

1997.11.12 AD at 04:10:18 PST

# Example 12.1 Displaying a Clock

Objective: Display current time based on the specified locale and time zone. The program can run as applet or application. The language, country, and time zone are passed to the program as command-line arguments like this:

javaw CurrentTimeApplet en US CST

<u>CurrentTimeApplet</u>

Run as Application

Run as Applet

# Example 12.2 Displaying a Calendar

Objective: Display the calendar based on the specified locale. The user can specify a locale from a combo box that consists of a list of all the available locales supported by the system.

**CalendarApplet** 

Run as Application

Run as Applet

# **Formatting Numbers**

Formatting numbers as currency or percentages is highly locale dependent.

For example, number 5000.50 is displayed as \$5,000.50 in the US currency, but the same number is displayed as 5 000,50 F in the French currency.

## The NumberFormat Class

use one of the factory class methods to get a formatter.

Use getInstance() or getNumberInstance()
to get the normal number format.

Use getCurrencyInstance() to get the currency number format.

Use getPercentInstance() to get a format for displaying percentages. With this format, a fraction like 0.53 is displayed as 53%.

# The NumberFormat Class (cont.)

For example, to display a number in percentages, you can use the following code to create a formatter for the given locale.

NumberFormat percForm =
NumberFormat.getPercentInstance(locale);

You can then use <u>percForm</u> to format a number into a string like this:

String s = percForm.format(0.075);

# Example 12.3 Formatting Numbers

Objective: This example creates a mortgage calculator similar to the one in Example 10.1, "Using Applets." This new mortgage calculator allows the user to choose locales, and displays numbers in locale-sensitive format.

NumberFormattingDemo

Run as Application

Run as Applet

# Resource Bundles (Optional)

A *resource bundle* is a Java class file or a text file that provides locale-specific information. This information can be accessed by Java programs dynamically.

When your program needs a locale-specific resource, a message string for example, your program can load the string from the resource bundle that is appropriate for the desired locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.

# Example 12.4 Using Resource Bundles

 Objective: This example modifies the NumberFormattingDemo program in Example 12.3 to display messages, title, and button labels in English and French languages.

