Chapter 36 Internationalization

Objectives

✦ To describe Java's internationalization features (§36.1).

✦ To construct a locale with language, country, and variant (§36.2).

✦ To display date and time based on locale (§36.3).

✦ To display numbers, currencies, and percentages based on locale (§36.4).

✦ To develop applications for international audiences using resource bundles (§36.5).

✦ To specify encoding schemes for text I/O (§36.6).

Java's International Support

1. Use *Unicode*

2. Provide the Locale class to encapsulate information about a specific locale. A <u>Locale</u> object determines how locale-sensitive information, such as date, time, and number, is displayed, and how locale-sensitive operations, such as sorting strings, are performed.

3. Use the ResourceBundle class to separate locale-specific information such as status messages and the GUI component labels from the program. The information is stored outside the source code and can be accessed and loaded dynamically at runtime from a <u>ResourceBundle</u>, rather than hard-coded into the program.
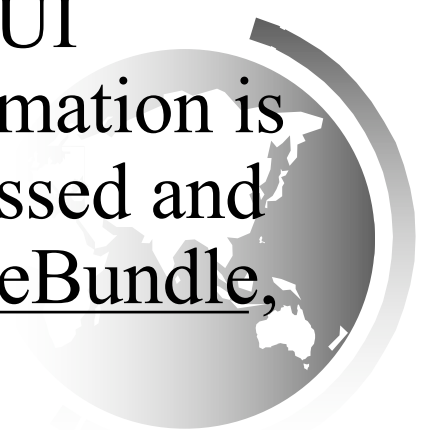
The `Locale` Class

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.

java.util.Local	
+Locale(language: String)	Constructs a locale from a language code.
+Locale(language: String, country: String)	Constructs a locale from language and country codes.
+Locale(language: String, country: String, variant: String)	Construct a locale from language, country, and variant codes.
+getCountry(): String	Returns the country/region code for this locale.
+getLanguage(): String	Returns the language code for this locale.
+getVariant(): String	Returns the variant code for this locale.
+getDefault(): Locale	Gets the default locale on the machine.
+getDisplayCountry(): String	Returns the name of the country as expressed in the current locale.
+getDisplayLanguage(): String	Returns the name of the language as expressed in the current locale.
+getDisplayName(): String	Returns the name for the locale. For example, the name is <u>Chinese (China)</u> for the locale <u>Locale.CHINA</u>.
+getDisplayVariant(): String	Returns the name for the locale's variant if exists.

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");

Locale.CANADA

Locale.CANADA_FRENCH
```

The Locale-Sensitive Operations

An operation that requires a Locale to perform its task is called *locale-sensitive*. Displaying a number as a date or time, for example, is a locale-sensitive operation; the number should be formatted according to the customs and conventions of the user's locale.

Several classes in the Java class libraries contain locale-sensitive methods. Date, Calendar, DateFormat, and NumberFormat, for example, are locale-sensitive. All the locale-sensitive classes contain a static method, getAvailableLocales(), which returns an array of the locales they support. For example,

```
Locale[] availableLocales = Calendar.getAvailableLocales();
```
returns all the locales for which calendars are installed.

Processing Date and Time

java.util.Date
   Introduced in Chapter 9.

java.util.Calendar and java.util.GregorianCalendar
   Introduced in Chapter 13.

Different locales have different conventions for displaying date and time. Should the year, month, or day be displayed first? Should slashes, periods, or colons be used to separate fields of the date? What are the names of the months in the language? The java.text.DateFormat class can be used to format date and time in a locale-sensitive way for display to the user.

The `TimeZone` Class

TimeZone represents a time zone offset and also figures out daylight savings. To get a TimeZone object for a specified time zone ID, use TimeZone.getTimeZone(id). To set a time zone in a Calendar object, use the setTimeZone method with a time zone ID. For example, cal.setTimeZone(TimeZone.getTimeZone("CST")) sets the time zone to Central Standard Time. To find all the available time zones supported in Java, use the static method getAvailableIDs() in the TimeZone class. In general, the international time zone ID is a string in the form of continent/city like Europe/Berlin, Asia/Taipei, and America/Washington. You can also use the static method getDefault() in the TimeZone class to obtain the default time zone on the host machine.

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 `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`.

java.text.DateFormat	
+format(date: Date): String	Formats a Date into a date/time string.
+getDateInstance(): DateFormat	Gets the date formatter with the default formatting style for the default locale.
+getDateInstance(dateStyle: int): DateFormat	Gets the date formatter with the given formatting style for the default locale.
+getDateInstance(dateStyle: int, aLocale: Locale): DateFormat	Gets the date formatter with the given formatting style for the given locale.
+getDateTimeInstance(): DateFormat	Gets the date and time formatter with the default formatting style for the default locale.
+getDateTimeInstance(dateStyle: int, timeStyle: int): DateFormat	Gets the date and time formatter with the given date and time formatting styles for the default locale.
+getDateTimeInstance(dateStyle: int, timeStyle: int, aLocale: Locale): DateFormat	Gets the date and time formatter with the given formatting styles for the given locale.
+getInstance(): DateFormat	Get a default date and time formatter that uses the SHORT style for both the date and the time.

`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 user-defined 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");
```
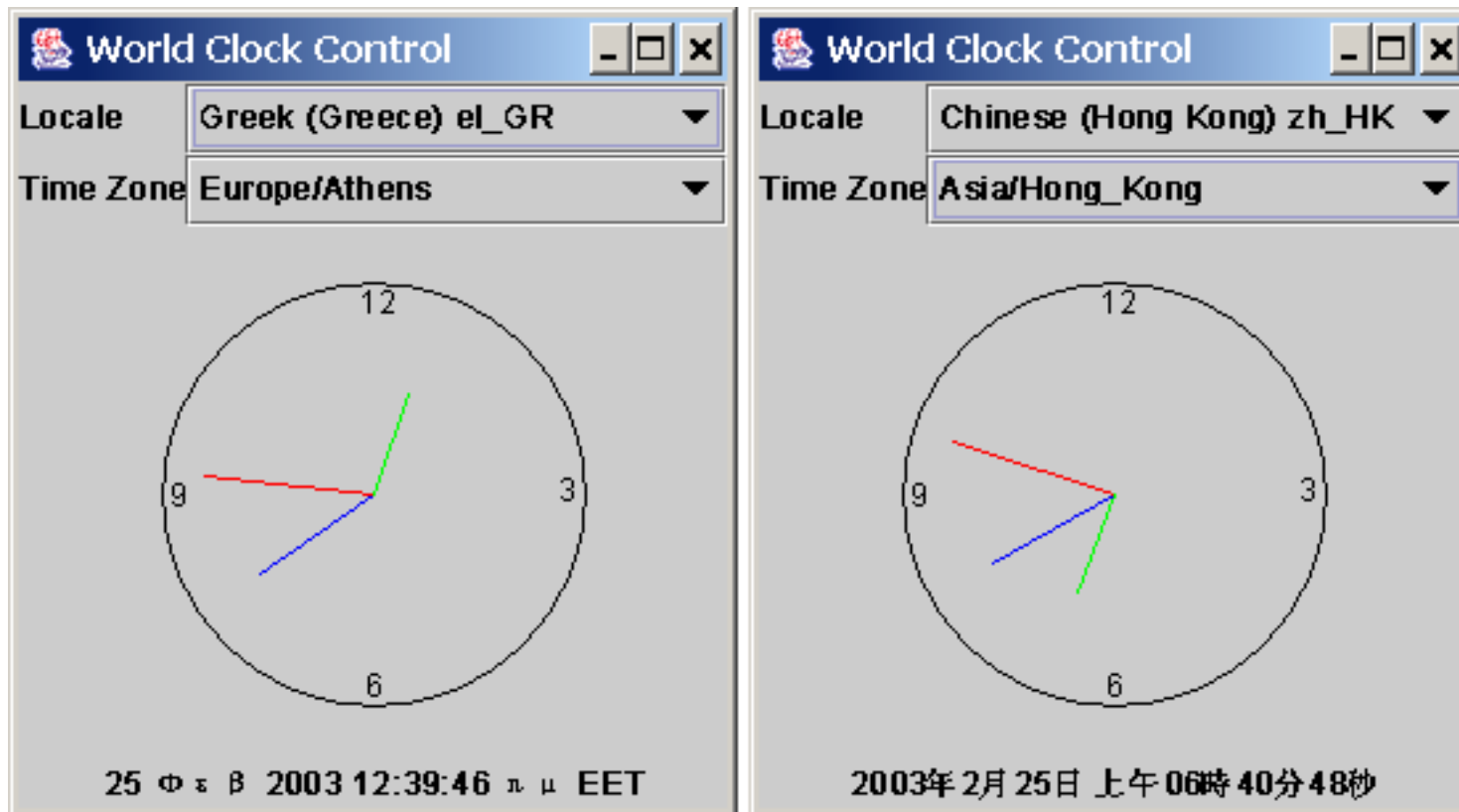
```
1997.11.12 AD at 04:10:18 PST
```
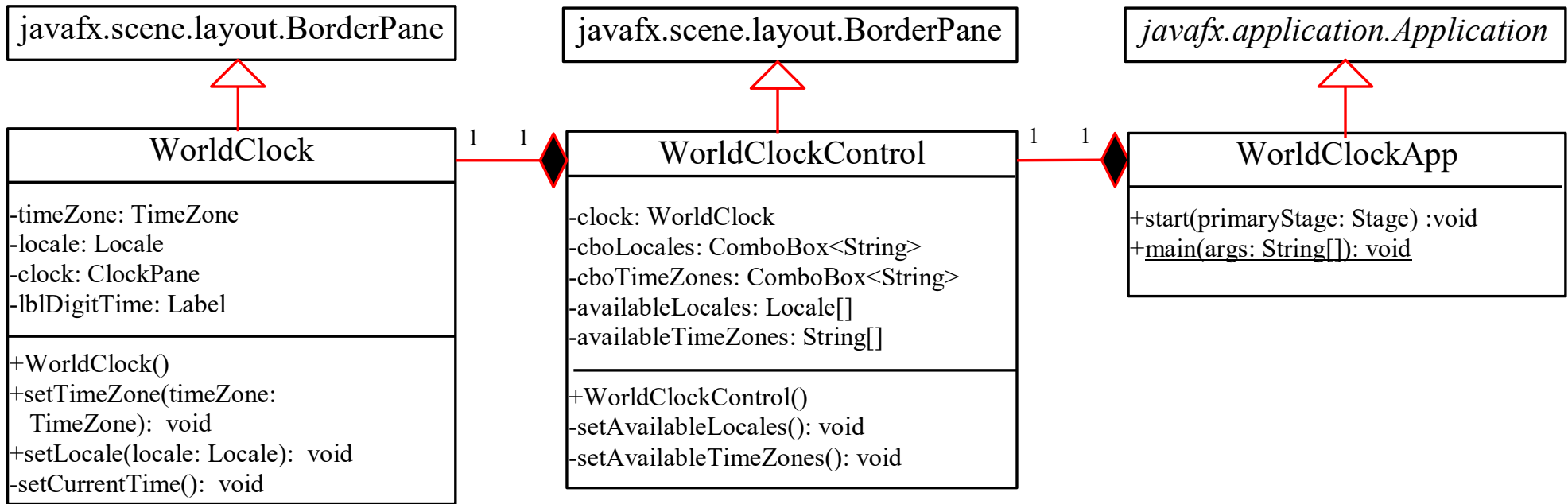
Example:
Displaying an International Clock

✦ Objective: Write a program that displays a clock to show the current time based on the specified locale and time zone. The locale and time zone are selected from the combo boxes that contain the available locales and time zones in the system.

Example, cont.

Example, cont.

javafx.scene.layout.BorderPane

javafx.scene.layout.BorderPane

javafx.application.Application

WorldClock
-timeZone: TimeZone -locale: Locale -clock: ClockPane -lblDigitTime: Label
+WorldClock() +setTimeZone(timeZone: TimeZone): void +setLocale(locale: Locale): void -setCurrentTime(): void

WorldClockControl
-clock: WorldClock -cboLocales: ComboBox\<String> -cboTimeZones: ComboBox\<String> -availableLocales: Locale[] -availableTimeZones: String[]
+WorldClockControl() -setAvailableLocales(): void -setAvailableTimeZones(): void

WorldClockApp
+start(primaryStage: Stage) :void +main(args: String[]): void

1   1

1   1

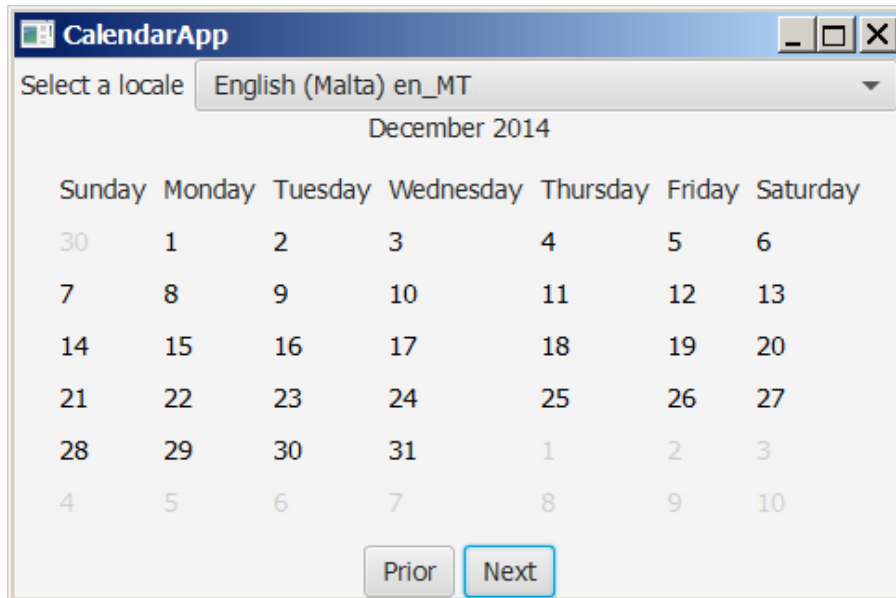| WorldClock	WorldClockControl	WorldClockApp

Example:
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.
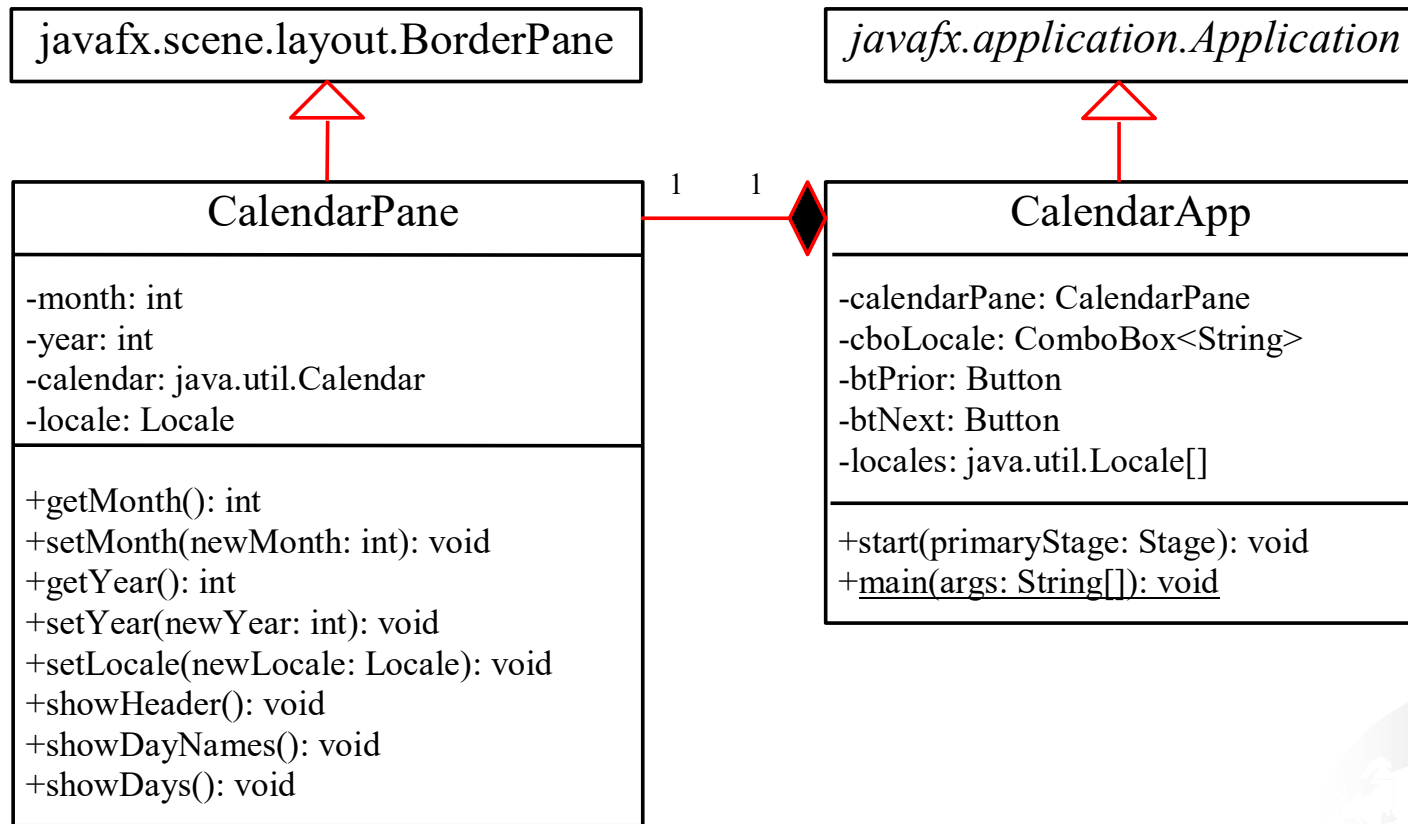
Example, cont.

CalendarApp

Example, cont.

javafx.scene.layout.BorderPane

javafx.application.Application

CalendarPane
-month: int -year: int -calendar: java.util.Calendar -locale: Locale
+getMonth(): int +setMonth(newMonth: int): void +getYear(): int +setYear(newYear: int): void +setLocale(newLocale: Locale): void +showHeader(): void +showDayNames(): void +showDays(): void

1   1

CalendarApp
-calendarPane: CalendarPane -cboLocale: ComboBox<String> -btPrior: Button -btNext: Button -locales: java.util.Locale[]
+start(primaryStage: Stage): void +main(args: String[]): void

CalendarPane

CalendarApp

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.

java.text.NumberFormat	
+getInstance(): NumberFormat	Returns the default number format for the current default locale.
+getInstance(locale: Locale): NumberFormat	Returns the default number format for the specified locale.
+getIntegerInstance(): NumberFormat	Returns an integer number format for the current default locale.
+getIntegerInstance(locale: Locale): NumberFormat	Returns an integer number format for the specified locale.
+getCurrencyInstance(): NumberFormat	Returns a currency format for the current default locale.
+getNumberInstance(): NumberFormat	Returns a general-purpose number format for the current default locale.
+getNumberInstance(locale: Locale): NumberFormat	Returns a general-purpose number format for the specified locale.
+getPercentInstance(): NumberFormat	Returns a percentage format for the current default locale.
+getPercentInstance(locale: Locale): NumberFormat	Returns a percentage format for the specified locale.
+format (number: double): String	Formats a floating-point number.
+format (number: long): String	Formats an integer.
+getMaximumFractionDigits(): int	Returns the maximum number of allowed fraction digits.
+setMaximumFractionDigits(newValue: int): void	Sets the maximum number of allowed fraction digits.
+getMinimumFractionDigits(): int	Returns the minimum number of allowed fraction digits.
+setMinimumFractionDigits(newValue: int): void	Sets the minimum number of allowed fraction digits.
+getMaximumIntegerDigits(): int	Returns the maximum number of allowed integer digits in a fraction number.
+setMaximumIntegerDigits(newValue: int): void	Sets the maximum number of allowed integer digits in a fraction number.
+getMinimumIntegerDigits(): int	Returns the minimum number of allowed integer digits in a fraction number.
+setMinimumIntegerDigits(newValue: int): void	Sets the minimum number of allowed integer digits in a fraction number.
+isGroupingUsed(): Boolean	Returns true if grouping is used in this format. For example, in the English locale, with grouping on, the number 1234567 might be formatted as "1,234,567".
+setGroupingUsed(newValue: boolean): void	Set whether or not grouping will be used in this format.
+parse(String source): Number	Parses string into a number.
+getAvailableLocales(): Locale[]	Gets the set of Locales for which NumberFormats are installed.

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:
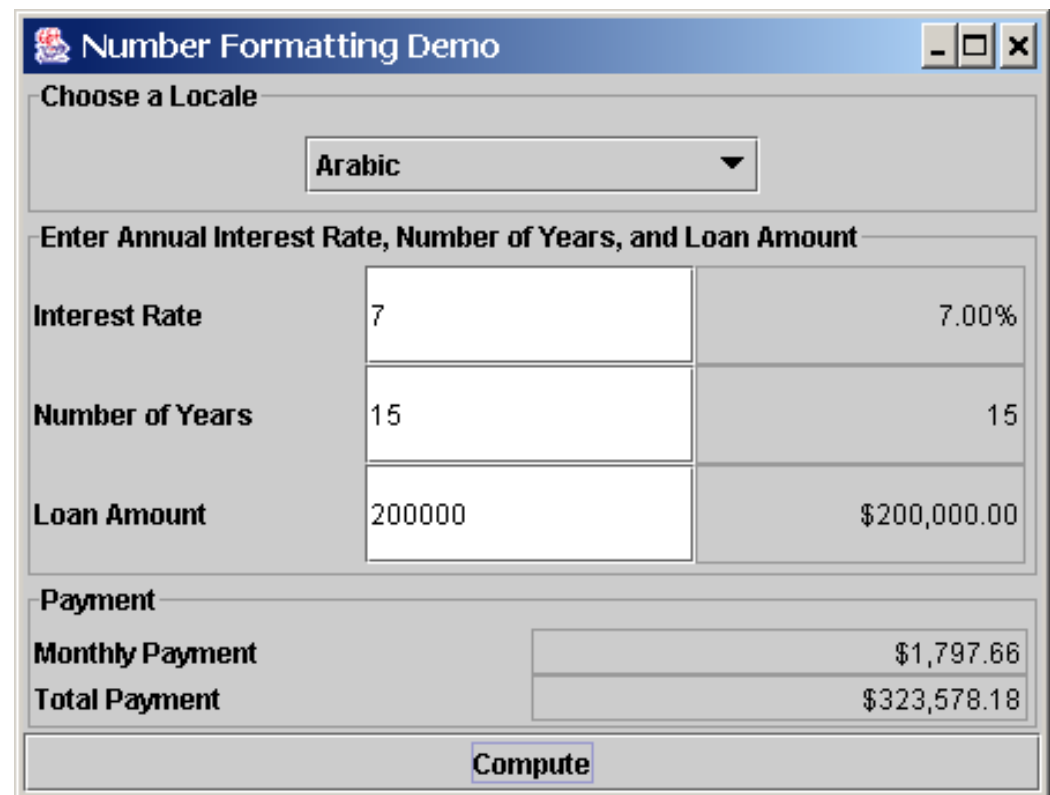
String s = percForm.format(0.075);

Example: Formatting Numbers

Objective: This example creates a loan calculator similar to the one in Listing 15.6. This new loan calculator allows the user to choose locales, and displays numbers in locale-sensitive format.

Number Formatting Demo

Choose a Locale

Arabic ▼

Enter Annual Interest Rate, Number of Years, and Loan Amount

Interest Rate	7	7.00%
Number of Years	15	15
Loan Amount	200000	$200,000.00

Payment

| Monthly Payment | | $1,797.66 |
| Total Payment | | $323,578.18 |

Compute

NumberFormatDemo

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: Using Resource Bundles

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

Example, cont.

ResourceBundleDemo

Liang, Introduction to Java Programming and Data Structures, Twelfth Edition, (c) 2020
Pearson Education, Inc. All rights reserved.