# Internationalization

The lessons in this trail teach you how to internationalize Java applications. Internationalized applications are easy to tailor to the customs and languages of end users around the world.

**Introduction**

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first "i" and the last "n."

An internationalized program has the following characteristics:

* With the addition of localized data, the same executable can run worldwide.
* 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.
* 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** is the process of adapting software for a specific region or language by adding locale-specific components and translating text. The term localization is often abbreviated as l10n, because there are 10 letters between the "l" and the "n."

# Before Internationalization

Suppose that you've written a program that displays three messages, as follows:

public class NotI18N {

static public void main(String[] args) {

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

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

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

}

}

You've decided that this program needs to display these same messages for people living in France and Germany. Unfortunately your programming staff is not multilingual, so you'll need help translating the messages into French and German. Since the translators aren't programmers, you'll have to move the messages out of the source code and into text files that the translators can edit. Also, the program must be flexible enough so that it can display the messages in other languages, but right now no one knows what those languages will be.

It looks like the program needs to be internationalized.

# After Internationalization

The source code for the internationalized program follows. Notice that the text of the messages is not included in the software 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"));

}

}

To compile and run this program, you need these source files:

* I18NSample.java
* MessagesBundle.properties
* MessagesBundle\_de\_DE.properties
* MessagesBundle\_en\_US.properties
* MessagesBundle\_fr\_FR.properties

# Running the Sample Program

The internationalized program is flexible; it allows the end user to specify a language and a country on the command line. In the following example the language code is fr (French) and the country code is FR (France), so the program displays the messages in French:

% java I18NSample fr FR

Bonjour.

Comment allez-vous?

Au revoir.

In the next example the language code is en (English) and the country code is US (United States) so the program displays the messages in English:

% java I18NSample en US

Hello.

How are you?

Goodbye.