In diesem Kapitel:

Internationalisierung und Lokalisierung

Nobody can be exactly like me.
Sometinmes even I have trouble doing it.
- Tallulah Bankhead

Das Credo "Write once, run anywhere" bedeutet, dass Ihre Java Programm auf mehreren Plattformen und unterschiedlichen geografischen und Sprach- Regionen lauffähig sein sollte. Mit etwas Aufwand können Sie Programme so schreiben, dass sie sanft an sprachliche Regionen und Zeitzonen angepasst werden können. Falls Sie Ihr Programm so auslegen, dass es in unterschiedlichen Sprachen einsetzbar ist, schaffen Sie die Grundlage für die *Internationalisierung* Ihrer Programme. Java bietet Ihnen dazu unterschiedliche Hilfsmittel an. Falls Sie Ihr Programm an die lokalen Gegebenheiten anpassen, also die Meldungen in einer passenden Sprache erscheinen, schaffen Sie die Grundlage für eine *Lokalisierung* Ihrer Programme.

Das erste Hilfsmittel, welches fix in Java eingebaut ist, ist Unicode, also die Darstellung der Zeichenketten in einem globalen Kodierungsschema. Die Zeichenketten müssen zwar immer noch in die jeweilige Sprache übersetzt werden, aber Unicode ist eine solide Basis dafür.

Die Verknüpfung von Internationalisierung und Lokalisierung ist das *locale*, welches einen "Platz" definiert. Ein Platz kann eine Sprache, eine Kultur oder ein Land sein - alles was einem Kunden unseren Programme angepasst werden muss (aus der Sicht der Benutzersteuerung). Jedes Programm besitzt ein Standard locale, eine bevorzugte Einstellung. In Java sind die dazugehörigen Einstellungen in der Klasse `Locale` im Package `java.util` zusammengefasst.

Zusätzliche Hilfsprogramme helfen, in Zusammenarbeit mit den Locale Objekten Ihr Programm lokalen Gegebenheiten anzupassen. Ein Standardmuster für die Programmierung international einsetzbarer Software besteht darin, die Methoden zu überladen. Je nach Sprache werden die länderspezifischen, besser locale-spezifischen, Methoden und damit die Texte eingesetzt. Eine andere Variante besteht darin, mit locale-spezifischen Unterklassen zu arbeiten.

Wir werden sehen, wie diese Mechanismen arbeiten, sobald wir konkrete Probleme, beispielsweise den Kalender, anschauen.

1.1. Ein Beispiel

Falls Sie noch nie ein Java Programm an internationale Anforderungen angepasst haben, ist es am einfachsten, mit einem Beispiel zu beginnen. Dabei wollen wir versuchen, einen einfachen Text in unterschiedlichen Sprachen auszugeben. Dabei lernen Sie an einem Beispiel, was ein Locale und was ResourceBundle Objekte sind, wie Sie diese einsetzen können und wie diese mit Propertydateien zusammenarbeiten.

1.1.1. Vor der Internationalisierung

Annahme: Sie haben ein einfaches Programm geschrieben, welches drei Meldungen ausgibt:

```
package einführendesbeispiel;

/**
 * Title:        Internationalisierung
 * Description:  AUsgangspunkt - ein lokales Programm
 * Copyright:    Copyright (c) 2000
* @version 1.0
 */

public class GoGlobal {
    static public void main(String[] args) {

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

Da Sie dieses Programm globale vermarkten möchten, Sie sehen echte Marktchancen dafür, wollen Sie an diesem Beispiel die Geheimnisse der Internationalisierung in Java kennen zu lernen.

Leider ist Ihr Programm noch schlecht auf den internationalen Markt vorbereitet und Ihr Kernprogrammierteam spricht nur eine Sprache. Sie wollen als erstes Frankreich und Deutschland als Märkte angehen und entsprechende Versionen erarbeiten. Da Ihre Programmierer nur Schweizerdeutsch können, sehen Sie sich gezwungen, den Text aus dem Programm herauszuholen, in eine Textdatei, welche von einem Übersetzer bearbeitet werden kann, speziell bei diesem komplexen Text.

1.1.2. Nach der Internationalisierung

Nun schauen wir uns die internationalisierte Version des Programms an. Beachten Sie, dass der Text aus dem Programm entfernt wurde:

```
package einführendesbeispiel;

/**
 * Title:        Internationalisierung
 * Description:  AUsgangspunkt - ein lokales Programm
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

import java.util.*;

public class BeGlobal {

   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"));
   }
}
```

Jetzt haben wir den Text in unterschiedliche, sprachspezifische Dateien ausgelagert.
Zusätzlich haben wir jetzt folgende Dateien:

MessagesBundle.properties

```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

MessagesBundle_en_US.properties
```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

MessagesBundle_de_DE.properties
```
greetings = Hallo.
farewell = Tschüß.
inquiry = Wie geht's?
```

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

1.1.2.1. Starten des Beispielprogramms

Das internationale Programm ist sehr flexibel und es gestattet es dem Benutzer seine Sprache zu spezifizieren, mittels einer Länderangabe beim Programmstart. Die Länderangabe sieht so aus:

Sprache	Land
fr	FR
en	US
de	DE

Der Programmaufruf ist

```
java BeGlobal <Sprachcode> <Ländercode>
```

Beispiel:

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

Entsprechend können Sie das Programm für Deustchland oder englisch (US) starten.

Nachdem wir das Ergebnis gesehen haben, wollen wir nun die internationale Version schrittweise erarbeiten.

1.1.3. Internationalisieren des Beispielprogramms

Beim Betrachten der internationalen Version sehen Sie, dass die englischen Texte entfernt wurden. Da die Meldungen nun nicht mehr fest eingebaut sind und der Sprachcode zur Laufzeit eingegeben werden kann, können Sie ein und dasselben Programm weltweit vertreiben. Bei der Lokalisierung braucht der Programmcode nicht angepasst zu werden. Das Programm *ist* international.

Vielleicht wundern Sie sich, was mit dem Text geschehen ist oder was der Sprach- und der Ländercode bedeuten.

Mit diesen Fragen beschäftigen wir uns jetzt.

1.1.3.1. Kreieren der Properties Dateien

Eine Propertiesdatei speichert Informationen über die Charakteristiken eines Programms oder einer Umgebung. Eine Propertiesdatei ist reiner Text. Sie können eine solche Datei mit jedem Texteditor kreieren.

In unserem Beispiel speichern die Propertiesdateien den übersetzten Text, die Meldungen, welche angezeigt werden. Vor der Internationalisierung war der die englische Version des Textes fest im Programm eingebaut.

Es gibt jeweils eine Standard `MessageBundle.properties` Datei. In unserem Fall enthält diese den Text:

```
greetings = Hello
farewell = Goodbye
inquiry = How are you?
```

Nachdem wir den Text ausgelagert haben, können wir ihn auch in andere Sprachen übersetzen ohne dass Änderungen am Programm nötig werden.

Die französische Übersetzung finden Sie in der Datei `MessagesBundle_fr_FR.properties`. Diese enthält folgende Zeilen:

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

Beachten Sie, dass der Text auf der linken Seite der selbe geblieben ist. Einzig auf der rechten Seite wurde der französische Text eingefügt.

Die Namen der Propertiesdateien ist sehr wichtig. Beispielsweise enthält der Name der französischen Datei `MessagesBundle_fr_FR.properties` den Sprachcode `fr` und den Ländercode `FR`.

Diese Codes werden auch eingesetzt, um ein Locale Objekt zu kreieren.

1.1.3.2. Definieren des Locale Objekts

Das Locale Objekt identifiziert eine bestimmte Sprache und dieses Land. Die folgende Anweisung definiert ein Locale für die Sprache Englisch und das Land USA:

```
aLocale = new Locale("en","US");
```

Für die andern Länder sieht die Definition des Locale Objekts folgendermassen aus:

```
caLocale = new Locale("fr","CA");
frLocale = new Locale("fr","FR");
```

Damit wird das Programm viel flexibler, da es die Ländereinstellung zur Laufzeit erhält:

```
String language = new String(args[0]);
String country = new String(args[1]);
currentLocale = new Locale(language, country);
```

Locale Objekte sind lediglich Bezeichner. Nach der Definition eines Locale werden diese an andere Objekte übergeben, welche länderspezifische Angaben verarbeiten, beispielsweise die Formattierung vom Datum oder der Darstellung numerischer Werte. Diese Objekte sind also spezifisch für die Gegend, das Umfeld und ihre Verhalten ändert sich je nach Locale Objekt.

Ein ResourceBundle Objekt ist so ein lokal abhängiges Objekt.

1.1.3.3. Kreieren eines ResourceBundle

ResourceBundle Objekte enthalten lokale Objekte. Man verwendet ResourceBundle Objekte, um lokale Daten, beispielsweise Text, zu isolieren. Im Beispielprogramm steht das ResourceBundle Objekt in Verbindung zu den Properties Dateien, welche die Textmeldungen enthalten, die wir anzeigen wollen.

Das ResourceBundle Objekt wird folgendermassen kreiert:

message = ResourceBundle.getBundle("MessagesBundle",currentLocale);

Die Argumente der `getBundle()` Methode identifiziert, welche Propertiesdatei benutzt wird. Das erste Argument bezeichnet die Familie der Propertiesdateien:

```
MessagesBundle_en_US.properties
MessagesBundle_fr_FR.properties
MessagesBundle_de_DE.properties
```

Das `Locale` Objekt ist das zweite Argument der `getBundle()` Methode. Es spezifiziert, welche `MessagesBundle` Datei benutzt wird. Beim Kreieren des `Locale` Objekts, wurden der Sprachcode und der Ländercode als Parameter verwendet. Der Länder- und der Sprachcode sind Teil des Propertiesdateinamens.

Alles was wir nun noch tun müssen, ist den länderspezifischen Text aus dem `ResourceBundle` zu lesen.

1.1.3.4. Lesen des Textes aus dem ResourceBundle

Die Propertiesdateien enthalten <Schlüssel, Wert> Paare. Die Werte bestehen aus dem übersetzten Text, den das Programm anzeigen wird. Um den Text aus dem `ResourceBundle` zu lesen, benötigen Sie den Schlüssel und die `getString()` Methode.

Um beispielsweise den begrüssenden Text zu bestimmen, verwenden Sie den Schlüssel "greetings":

```
String msg1 = messages.getString("greetings");
```

Wir könnten genausogut einen beliebigen Schlüssel verwenden, etwa msg1. Der Schlüssel ist der Teil, welcher im Programm hart kodiert ist. Falls der Übersetzer der Textdateien den Schlüssel verändert, kann der Wert nicht mehr gefunden werden.

1.1.3.5. Abschluss

Das ist alles zum ersten Beispiel! Sie haben es geschafft ein erfolgreiches lokales Programm zu globalisieren, internationalisieren und trotzdem zu lokalisieren.

Nun steht Ihrem internationalen Erfolg nichts mehr im Wege!

1.1.3.6. Checkliste

Viele Programme sind nicht auf den internationalen Einsatz ausgerichtet, wenn sie erstellt werden. Diese Programme wurden eventuell als Prototypen entwickelt. Daher ist es Ihre Aufgabe, ein bestehendes Programm zu internationalisieren.

Die geschieht in mehreren Schritten.

1.1.3.6.1. Identifizieren Sie kulturell definierte Daten

Textmeldungen sind ein Beispiel für kulturell veränderliche Daten. Aber es gibt auch andere Datentypen, welche regional angepasst werden müssen. Die folgende Liste zeigt Beispiele für kulturabhängige Daten:

- Meldungen
- Labels auf GUI Komponenten
- Online Help
- Sounds
- Farben
- Graphiken
- Icons
- Datum
- Uhrzeit
- Zahlen
- Währung
- Messeinheiten
- Telefonnummern
- Ehren- und persönliche Titel
- Postadresse
- Seitenlayout

1.1.3.6.2. Isolieren Sie übersetzbaren Text in Ressource Bündeln.

Die Übersetzung ist zeitaufwendig und teuer. Sie können diesen Aufwand reduzieren, indem Sie Texte isolieren und in ResourceBundle Objekten abspeichern.

Übersetzbarer Text umfasst insbesondere auch die Statusmeldung, Fehlermeldungen, Logdateieinträge und die Beschriftung der GUI Komponenten.

Dieser Text ist in Programmen, welche nur für lokalen Einsatz gedacht sind, in der Regel fest einprogrammiert.

Sie sollten beispielsweise die folgenden Programmzeilen bereinigen:

```
String buttonLabel = "OK";
...
JButton okButton = new JButton(buttonLabel);
```

1.1.3.6.3. Passen Sie zusammengesetzte Meldungen an

Zusammengesetzte Meldungen enthalten variable Daten. In der Meldung "Die Festplatte enthält 1000 Dateien" könnte die Zahl '1000' genausogut eine Variable sein. Sie können solche Meldungen nur sehr schwer übersetzen, da eine Aufteilung des Textes sprachabhängig ist:

```
Integer fileCount;
...
String diskStatus = "The disk contains " + fileCount.toString()
                    + " files.";
```

Eine Übersetzung dieser `diskStatus` Meldung ist kaum automatisch machbar, weil die Zahl je nach Sprache an unterschiedlichen Stellen stehen kann.

1.1.3.6.4. Formattieren von Zahlen und Währung

Falls Ihre Applikation Zahlen und Währungen anzeigt, müssen Sie diese auf eine lokalunabhängige Art und Weise darstellen. Das folgende Beispiel ist noch nicht internationalisiert, weil die Daten nicht sprachneutral angezeigt werden.

```
Double amount;
TextField amountField;
...
String displayAmount = amount.toString();
amountField.setText(displayAmount);
```

Die Lösung könnte so aussehen, dass Sie eine Methode definieren, welche die länderspezfischen Formattierungen vornimmt.

1.1.3.6.5. Formattierenvon Datum und Zeit

Datum und Zeitformate sind jeweils stark von der Region anhängig. Beispielsweise wäre das folgende Programm zu stark auf lokale Gegebenheiten ausgerichtet:

```
Date currentDate = new Date();
TextField dateField;
...
String dateString = currentDate.toString();
dateField.setText(dateString);
```

Falls Sie an Stelle einer fixen Darstellung mit den Klassen `Dates` und `Times` arbeiten, haben Sie dieses Problem nicht.

1.1.3.6.6. Verwenden Sie Unicode Zeicheneigenschaften

Im folgenden Programmfragment wird versucht, zu überprüfen, ob ein Zeichen ein Buchstabe ist:

```
char ch;
...
if ((ch >= 'a' && ch <= 'z') ||
    (ch >= 'A' && ch <= 'Z'))       // falsch!
```

Der obige Programmcode stimmt in Englisch. In Deutsch wird er schon nicht so ausgeführt, wie Sie es vermutlich erwarten: ü, Ü sind im Englischen nicht vorhanden.

Die gleiche Abfrage, aber korrekt und unter Ausnutzung von Unicode-Eigenschaften bzw. Methoden:

```
char ch;
...
if (Character.isLetter(ch))
```

1.1.3.6.7. Vergleichen Sie Zeichenketten korrekt

Beim Sortieren von Text werden Zeichenketten verglichen. Falls Text abgezeigt wird, sollten keine Entscheide auf Grund von Textvergleich gemacht werden.

Hier eine lokale Version:

```
String target;
String candidate;
...
if (target.equals(candidate)) {
...
if (target.compareTo(candidate) < 0) {
...
```

Die String.equals() und Stringcompare() Methoden benutzen binäre Vergelichsmethoden.Diese sind ineffizient in fast allen Sprachen. Sie sollten an Stelle des obigen Verfahrens besser mit dem Collator Objekt arbeiten (auf dieses werden wir noch zurück kommen).

1.1.3.6.8. Konvertieren Sie Nicht-Unicode Text

Zeichen weren in Java mittels Unicode kodiert. Falls Ihr Programm nicht-Unicode Text enthält, sollten Sie versuchen, den Text in Unicode umzuwandeln.

Wie man dies erledigen kann, besprechen wir weiter unten.

1.2. Locales

Nun befassen wir uns genauer mit der Definition un dem Einsatz der Locale Objekte, zuerst eher auf praktische Art und Weise, anschliessend fassen wir die Methoden der Klasse zusammen und beschreiben diese.

Ein internationalisiertes Programm kann Informationen rund um den Globus unterschiedlich anzeigen. Zum Beispiel kann ein Programm unterschiedliche Meldungen in Paris, Tokyo oder New York anzeigen. Falls Ihr Programm sehr fein unterscheidet, wird die Anzeige in London nicht die gleiche sein wie in New York, obschon beide englisch sind.

Wie werden diese lokalen Unterschiede erfasst und fetgehalten?
In Java geschieht dies mit Hilfe eines `Locale` Objekts.

Ein Locale Objekt identifiziert eine Kombination von Sprache und Region. Falls ein Objekt sein Verhalten gemäss dem Locale verändert, bezeichnet man es als locale-sensitive. Die NumberFormat Klasse ist locale-sensitive; das Format der Zahl, die zurückgegeben wird, hängt vom Locale ab. NumberFormat liefert beispielweise eine Zahl als 902 300 (Frankreich), oder 902.300 (Deutschland) oder 902,300 (United States). Locale Objekte sind nur Bezeichner. Die eigentliche Formatttierung müssen die locale-sensitiven Objekte erledigen.

1.2.1. Kreieren eines Locale

Um ein Locale Objekt zu kreieren müssen Sie typischerweise den Sprachcode und den ISO Ländercode angeben. Falls Sie französisch können und auch noch gerne Ihre Prototypen in französisch entwickeln, aber einKanada Fan sind, werden Sie eventuell folgende Kombination wählen:

```
aLocale = new Locale("fr", "CA");
```

Als nächstes betrachten wir die Beispiele für Englisch in den USA und in Great Britain:

```
bLocale = new Locale("en", "US");
cLocale = new Locale("en", "GB");
```

Das erste Argument ist der Sprachcode, zwei Kleinbuchstaben gemäss dem ISO Code ISO-639.[1] Die folgende Tabelle listet einige der gängigen Sprachcodes:

Sprach Code	**Beschreibung**
de	German
en	English
fr	French
ja	Japanese
jw	**Javanese**
ko	Korean
zh	Chinese

[1] http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt Sie können diese auch abfragen

Das zweite Argument des Locale Konstruktors ist der Ländercode. Dieser besteht aus zwei Grossbuchstaben und entspricht ISO-3166[2]. Die folgende Liste listet einige der gängigen Ländercodes.

Ländercode	**Beschreibung**
CN	China
DE	Germany
FR	France
IN	India
US	United States

Falls Sie eine weitere Unterscheidung der Locale Objekte benötigen, können Sie ein selbst definiertes drittes Merkmal aufführen. Typischerweise unterscheiden Sie damit die Plattform, auf der das System laufen muss.

```
xLocale = new Locale("de", "DE", "UNIX");
yLocale = new Locale("de", "DE", "WINDOWS");
```

Dieser dritte Parameter entspricht keinem Standard. Die Unterscheidung ist willkürlich und spezifisch für Ihre Anwendung. Falls Sie Variantencodes einsetzen, wird nur Ihre Applikation diese kennen.

Ländercode und Variantencode sind optional. Falls Sie den Ländercode weglassen, spezifizieren Sie einen null, leeren String. Hier ein Beispiel für ein englisches Locale Objekt:

```
enLocale = new Locale("en", "");
```

Die Klasse stellt einige Konstanten zur Verfügung, mit denen Sie Sprachcodes oder Ländercodes überprüfen können. Beispielsweise können Sie ein japanisches Locale Objekt kreieren, indem Sie JAPANESE oder JAPAN als Konstanten verwenden.

Es gibt allerdings eine Finesse: die zwei folgenden lokalen Objekte sind äquivalent

```
j1Locale = Locale.JAPAN;
j2Locale = new Locale("ja", "JP");
```

Aber wenn Sie ein Locale einer Sprachkonstante gleich setzen, bleibt der Länderteil des Locale Objekts undefiniert (man spricht ja auch in mehreren Ländern die selbe Sprache). Das Schema ist offensichtlich nicht konsequent, da man auch auch bei Angabe des Landes nicht auf die Sprache schliessen kann (in Java scheinbar schon):

```
j3Locale = Locale.JAPANESE;
j4Locale = new Locale("ja", "");
```

[2] http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html Sie können diese auch abfragen

1.2.2. Identifzieren verfügbarer Locales

Sie können beliebige Kombinationen vonLändern und Sprachcodes definieren. Die Frage ist nur, ob das auch Sinn macht. Ein Locale Objekt ist lediglich ein Bezeichner. Das Locale Objekt wird an andere Objekte weitergegeben und diese erledigen dann die Arbeit. Diese anderen Objekte, die man als locale-sensitive Objekte bezeichnet, wissen nicht, wie sie auf diese Kombinationen reagieren müssen.

Sie können in einem locale-sensitive Objekt abfragen, welche Locales dieses Objekt interpretieren kann. Beispielsweise können Sie herausfinden, welche Locals die `DateFormat` Klasse versteht und unterstützt, indem Sie die `getAvailableLocales()` Methode aufrufen:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @version 1.0
 */

import java.util.*;
import java.text.*;

public class VerfuegbareLocals {
    static public void main(String[] args) {
      Locale list[] = DateFormat.getAvailableLocales();
      for (int i = 0; i < list.length; i++) {
          System.out.println(list[i].toString());
      }
    }
}
```

Die Ausgabe der toString() Methode liefert eine Zeichenkette, bei der Sprachcode und Ländercode durch '_' verbunden sind.

```
ar_EG
be_BY
bg_BG
ca_ES
cs_CZ
da_DK
de_DE
de_CH
.
fr_CH
.
it_CH
.
```

Die obige Ausgabe ist nicht so benutzerfreundlich. Aber in Java steht natürlich eine Methode zur Verfügung, welche Ihnen diese Codes gleich in lesbarem Text ausgibt. Mit der Methode

`getDisplayName()` an Stelle von `toString()` erhaltenSie eine Liste mit unter anderem folgenden Ausgaben, unvollständig wiedergegeben:

```
Englisch (Vereinigte Staaten)
Arabic (Oman)
Arabic (Qatar)
Byelorussian (Belarus)
Catalan (Spanien)
Catalan (Spanien,Euro)
Dänisch (Dänemark)
Deutsch
Deutsch (Österreich)
Deutsch (Österreich,Euro)
Deutsch (Schweiz)
Deutsch (Deutschland)
Deutsch (Deutschland,Euro)
Deutsch (Luxembourg)
Deutsch (Luxembourg,Euro)
Englisch (Australia)
Englisch (Kanada)
Englisch (Vereinigtes Königreich)
Englisch (Ireland)
Englisch (Ireland,Euro)
Englisch (New Zealand)
Englisch (South Africa)
French
French (Belgien)
French (Belgien,Euro)
French (Kanada)
French (Schweiz)
French (France)
French (France,Euro)
French (Luxembourg)
French (Luxembourg,Euro)
Italienisch
Italienisch (Schweiz)
Italienisch (Italien)
Italienisch (Italien,Euro)
Latvian (Lettish)
Latvian (Lettish) (Latvia)
Holländisch
Holländisch (Belgien)
Holländisch (Belgien,Euro)
Holländisch (Niederlande)
Holländisch (Niederlande,Euro)
```

Witzig finde ich die wilde Mischung von lokaler und englischer Bezeichnung der Sprache und der Länder.

1.2.3. Der Gültigkeitsbereich eines Locals

Die Java Plattform lässt es offen, ob Sie ein oder mehrere Locals in Ihrem Programm einsetzen. Falls Sie wollen, können Sie in Ihrem Programm unterschiedliche Locals einsetzen und im Laufe des Programms auch verändern, im Extremfall bei jedem local-sensitiven Objekt- Damit können Sie mehrsprachige Applikationen entwickeln, Applikationen also, welche die Informationen in mehreren Sprachen anzeigen können.

Falls Sie entweder zu faul sind mit Locals zu arbeiten, oder vergessen haben, wie das funktioniert, dann kommt Ihnen die JVM entgegen: beim Start der JVM wird der Sprachcode und der Ländercode bestimmt und Meldungen, sofern vorhanden, mehrsprachig, in der systemseitig vorgegebenen Sprache aus.

Sie können die Standardeinstellung auch leicht bestimmen, indem Sie diese mit `Locale.getDefault()` abfragen.

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
* @version 1.0
 */

import java.util.*;
public class StandardLocale {

  public StandardLocale() {
    System.out.println("Standard Locale: "+Locale.getDefault());
  }
  public static void main(String[] args) {
    StandardLocale standardLocale1 = new StandardLocale();
  }
}
```

```
Standard Locale: de_CH
```

Turbulent wird es, wenn Sie verteilte Anwendungen rund um den Globus am Laufen haben und jeweils in der lokalen Sprache die Ausgabe machen möchten.

Als Regel gilt:
in heterogenen Sprachumgebungen sollte die Kommunikation zwischen Clients und Servern neutral geschehen. Die Clients sind für die korrekte Darstellung aller lokalen Gegebenheiten zuständig. Damit wird der Server und die Kommunikation entlastet.

Falls der Server die lokalen Versionen auch speichern muss, bleibt Ihnen keine Wahl!

1.2.4. Die Locale Klasse

Die java.util.Locale Objekte, die wir in den Beispielen gesehen haben, beschreiben lokale Eigenschaften.

Die Klasse besitzt zwei Konstruktoren:

`public Locale(String language, String country, String variant)`
: Die Sprache wird dabei gemäss ISO 639, das Land gemäss ISO 3166 festgelegt("KY" = Cayman Island, für alle Fälle). Als Variante kann man beispielsweise das Betriebsystem angeben ("MAC", "POSIX", ...). Falls Sie eine Angabe weglassen wollen, können Sie einfach einen leeren String verwenden.

`public Locale(String language, String country)`
: ist äquivalent zu Locale(language, country, "");

Die Klasse definiert auch statische Locale Objekte für gängige Locals, beispielsweise KOREA, CANADA_FRENCH, TRADITIONAL_CHINESE. Es wird Ihnen auffallen, dass die Sprache der Texte nicht immer konsequent eingehalten wird. Die folgenden Beispiele werden Ihnen dieses wiederholt zeigen.

Die folgende Diskussion der Klassen im Unfeld von Text und Internationalisierung ist unvollständig. Es gibt sehr viele Klassen und Hilfsklassen, die Java Ihnen zur Verfügung stellt und wir stellen Ihnen nur einen kleinen Teil davon vor.

Die folgenden Klassen beschreiben zuerst die grundlegenden Klassen, wie ResourceBundles. Die anschliessende Auswahl an Klassen für die Formattierung und Zerlegung von Texten und Sätzen zeigt Ihnen nur exemplarisch was machbar ist.

1.3. Isolierung Locale-spezifischer Daten - ResourceBundle

Locale-spezifische Daten müssen gemäss den Anforderungen der Benutzersprache und Region angepasst werden. Die Texte der Benutzerschnittstelle sind ein typisches Beispiel dafür. Beispielsweise muss ein `Cancel` Knopf in Deutschland mit `Abbrechen` beschriftet sein. Also werden Sie diese Texte kaum fest einprogrammieren wollen. Dies ist auch nicht nötig, sofern Sie die Informationen in einem `ResourceBundle` isolieren.

Im Folgenden befassen wir uns mit diesem Thema. Zuerst befassen wir uns eher mit den Konzepten, anschliessend mit der konkreten Umsetzung. `ResourceBundle` Objekte enthalten locale-spezifische Objekte. Falls Sie ein locale-spezifisches Objekt benötigen, beschaffen Sie es sich vom `ResourceBundle`.

Bevor Sie ein `ResourceBundle` Objekt kreieren, müssen Sie genau planen, welche Objekte in welcher Art und Weise von lokalen Anpassungen betroffen sind. Die gewünschten Anpassungen an die Objekte lassen sich in der Regel in Kategorien einteilen, in Objektkategorien, welche analoges Verhalten zeigen. Die einzelnen Kategorien werden auf die `ResourceBundle` abgebildet.

Fall Ihr Programm einfache Zeichenketten enthält, welche in mehrere Sprachen übersetzt werden müssen, können Sie diese auch als Properties definieren und in einem `PropertyResourceBundle` abspeichern. Dieses wird auf Property Dateien abgebildet, also einfache Textdateien, welche Sie mit jedem Texteditor modifizieren und anlegen können.

Falls Ihre lokalen Texte besser mittels einer Liste gespeichert werden, können Sie ein `ListResourceBundle` definieren. Diese werden durch Java Klassen repräsentiert dh. in diesem Fall müssen Sie auch Java Programme zur Verfügung stellen.

1.3.1. Die ResourceBundle Klasse

1.3.1.1. Zusammenhang von Locales mit den ResourceBundles

Konzeptionell ist jedes ResourceBundle ein Set von zusammenhängenden Unterklssen, welche den selben Basisnamen haben. Die folgende Liste zeigtein solches Set. ButtonLabel ist der Basisnamen, die angehängten Namen zeigen den Sprachcode, Ländercode und einen allfälligen weiteren Unterscheidungscode:

```
ButtonLabel
ButtonLabel_de
ButtonLabel_en_GB
ButtonLabel_fr_CA_UNIX
```

Sie wählen das passende RessourceBundle mit der `ResourceBundle.getBundle()` Methode aus. Im folgenden Beispiel selektiert das ButtonLabelResourceBundle für Locale Objekte, welche französisch in Kanada und Unix unterstützen:

```
Locale currentLocale = new Locale("fr", "CA", "UNIX");
ResourceBundle introLabels =
    ResourceBundle.getBundle("ButtonLabel", currentLocale);
```

Falls das ResourceBundle Objekt für das betreffende Locale Objekt nicht gefunden wird, versucht die Methode das nächste Objekt zu finden, jenes ResourceBundle Objekt, welches möglichst viele der Auswahlkriterien erfüllt.

Beispiel:
falls unsere Button Beschriftung in französisch für Kanada und das Unx Betriebsystem nicht gefunden wird, kann das Standard Bundle, ..._en_US eingesetzt werden, sofern dies der Standard ist. Die Reihenfolge, mit der der Mechanismus eine Alternative sucht, sieht in diesem Fall folgendermassen aus:

```
ButtonLabel_fr_CA_UNIX
ButtonLabel_fr_CA
ButtonLabel_fr
ButtonLabel_en_US
ButtonLabel_en
ButtonLabel
```

Es werden also einige Klassen / Dateien gesucht bevor auf die Basisklasse (`ButtonLabel`) ausgewichen wird. Falls keine Klasse gefunden wird, kann eine `MissingResourceException` geworfen werden. Falls Sie das Werfen dieser Ausnahme vermeiden wollen, müssen Sie immer eine Basisklasse, eine ohne Ergänzung (`ButtonLabel`), bereitstellen.

1.3.1.2. ListResourceBundle und PropertyResourceBundle Unterklassen

Die abstrakte Klasse `ResourceBundle` besitzt zwei Unterklassen: `PropertyResourceBundle` und `ListResourceBundle`.

Ein PropertyResourceBundle wird durch eine Propertiesdatei unterstützt. Eine Propertiesdatei ist eine reine Textdatei, welche übersetzbaren Text enthält. Propertiesdateien sind Teil des Java Quellcodes und enthalten Werte für Zeichenkettenobjekte.

Falls Sie andere Objekttypen abspeichern müssen, werden Sie auf die ListResourceBundle Klasse ausweichen müssen, mit der Sie beliebige Objekte implementieren können. Die Objekte werden in diesem Fall in Form einer Liste angeordnet, einer Liste in Form eines Programms.

Die ResourceBundle Klasse ist flexibel: falls Sie in einer ersten Version lediglich Properties verwendet haben und später auf ListResourceBundles wechseln müssen, brauchen Sie Ihre Programme kaum anzupassen. Der Grund ist der, dass Sie mit Methodenaufrufen arbeiten. Diesen ist egal um welche Art Bundle es sich handelt.

```
ResourceBundle introLabels =
    ResourceBundle.getBundle("ButtonLabel", currentLocale);
```

1.3.1.3. Schlüssel-Werte Paare

ResourceBundle enthalten ein Datenfeld mit Schlüssel-Werte Paaren. Sie geben den Schlüssel an,eine Zeichenkette, und erhalten den gewünschten Wert des Schlüssels. Der Wert entspricht einem lokalen Objekt. Hier ein Beispiel für die Internationalisierung eines Ok und Cancel Buttons:

```
class ButtonLabel_en extends ListResourceBundle {
    // Englische version
    public Object[][] getContents() {
  return contents;
    }
    static final Object[][] contents = {
  {"OkKey", "OK"},
  {"CancelKey", "Cancel"},
    };
}
```

Um die Zeichenkette für den OK Button zu erhalten, würden Sie folgenden Aufruf machen:

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

Die dazugehörigen Propertydateien bestehen aus Schlüssel / Werte Paaren, wie in folgendem Beispiel:

```
OkKey = OK
CancelKey = Cancel
```

1.3.1.4. Identifizieren von Locale spezifischen Objekten

Falls Ihre Anwendung eine Benutzerinterface besitzt, werden darin viele locale-spezifische Objekte vorhanden sein. Als erstes sollten sie Ihren Quellcode anschauen und die Objekte identifizieren, welche lokale Objekte enthalten.

Typischerweise wird Ihre Liste Instanzen folgender Klassen enthalten:

- String
- Image
- Color
- AudioClip

Nicht aufgeführt haben wir Zahlen, Zahlenformate, Datum oder Währung. Diese werden speziell formattiert. Lokale Einstellungen werden sich im Laufe der Programmausführung kaum verändern. Es geht also um Einstellungen, die während der Ausführung eines Programms konstant bleiben. Textfelder, die ein Benutzer täglich durch seine Eingabe verändern kann, sind nicht Teil dieser lokalen Objekte.

Die meisten Objekte, die Sie isolieren werden, sind Textobjekte, Strings. Aber nicht jedes String Objekt ist ein Locale. Oft werden in Protokollen Zeichenketten als Steuerbefehle mitgegeben. Diese sollen Sie stehen lassen: den GET, PUT und POST Befehl von HTTP sollten Sie nicht in LIEFERE, SCHREIBE, TRAGE_EIN übersetzen.

Bei Logdateien ist der Entscheid schwieriger und unklarer. Ein Log kann beispielsweise in verteilten Anwendungen durch Clients in allenLändern mitbenutzt werden. Daher muss man sich dort eher auf eine Standardsprache festlegen, vermutlich.

1.3.1.5. Organisieren von ResourceBundle Objekten

Sie können Ihre ResourceBundle Objekte sinnvoll zusammenfassen, sinnvoll heisst beispielsweise:
die GUI Labels für die Auftragserfassung wird in einem OrderLabelBundle zusammengefasst.

In der Regel ist es besser, nicht alle ResourceBundle in einen Topf zu schmeissen:

- Sie erhöhen damit die Wartbarkeit Ihrer Anwendung
- Sie reduzieren die Grösse der ResourceBundle Objekte, welche sonst nur schwer und langsam zu laden wären.
- Sie optimieren den Speicherbedarf, indem Sie nur jene Teile laden, die aktuell benötigt werden.

1.3.1.6. Unterstützung eines ResourceBundle mit Propertiesdateien

In diesem Teilabschnitt schauen wir uns an, wie man mit Properties ResourceBundle Objekte unterstützen kann:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;

public class ResourceBundleMitProperties {
   static void displayValue(Locale currentLocale, String key) {

      ResourceBundle labels =
         ResourceBundle.getBundle("LabelsBundle",currentLocale);
      String value  = labels.getString(key);
      System.out.println(
           "Locale = " + currentLocale.toString() + ", " +
           "key = " + key + ", " +
           "value = " + value);

   } // displayValue


   static void iterateKeys(Locale currentLocale) {

      ResourceBundle labels =
         ResourceBundle.getBundle("LabelsBundle",currentLocale);

      Enumeration bundleKeys = labels.getKeys();

      while (bundleKeys.hasMoreElements()) {
         String key = (String)bundleKeys.nextElement();
         String value  = labels.getString(key);
         System.out.println("Schlüssel = " + key + ", " +
           "Wert = " + value);
```

```
      }

   } // iterateKeys

   static public void main(String[] args) {
      Locale[] supportedLocales = {
         Locale.FRENCH,
         Locale.GERMAN,
         Locale.ENGLISH
      };
      for (int i = 0; i < supportedLocales.length; i ++) {
          displayValue(supportedLocales[i], "s2");
      }
      System.out.println();
      iterateKeys(supportedLocales[0]);
   } // main
} // class
```

Die Ausführung liefert Informationen zu verschiedenen Rechnerbestandteilen in drei Sprachen:

```
Locale = fr, key = s2, value = Disque dur
Locale = de, key = s2, value = Platte
Locale = en, key = s2, value = Platte

Schlüssel = s4, Wert = Clavier
Schlüssel = s3, Wert = Moniteur
Schlüssel = s2, Wert = Disque dur
Schlüssel = s1, Wert = Ordinateur
```

Sobald wir eine Propertiesdatei LabelsBundle_en.properties hinzufügen, erhalten wir die treffendere Ausgabe:

```
Locale = fr, Schlüssel = s2, Wert = Disque dur
Locale = de, Schlüssel = s2, Wert = Platte
Locale = en, Schlüssel = s2, Wert = disk

Schlüssel = s4, Wert = Clavier
Schlüssel = s3, Wert = Moniteur
Schlüssel = s2, Wert = Disque dur
Schlüssel = s1, Wert = Ordinateur
```

Je nach Einstellung des Ländercodes erhalten Sie auch bereits früher eine englische Ausgabe: falls Ihr PC englisch als Standardsprache verwendet, kann es sein, dass das Programm die Standardpropertiesdatei wählt. In der stehen die Angaben in Englisch. Bei meinen Ländereinstellungen (CH, de) wird, falls keine englische Propertiesdatei vorliegt, einfach auf _de ausgewichen, in einer ersten Runde. Da diese Datei vorhanden ist, wird die Angabe aus dieser Datei angezeigt.

Wie geht man schrittweise vor?

1. kreieren Sie die Standard-Propertiesdatei
   Diese enthält neben den Schlüssel = Wert Paaren eventuell auch Kommentare, die mit einem # beginnen.

```
# Kommentar: dies ist die Standard Propertiesdatei
s1 = computer
s2 = disk
s3 = monitor
s4 = keyboard
```

2. kreieren Sie weitere Propertiesdateien, soviele wie Sie benötigen
3. spezifizieren Sie das Locale Objekt

```
Locale[] supportedLocales = {
    Locale.FRENCH,
    Locale.GERMAN,
    Locale.ENGLISH
};
```

   Sie dürfen nicht vergessen, zu jedem Locale eine entsprechende Datei zu kreieren.

4. kreieren Sie das ResourceBundle

```
ResourceBundle labels =
    ResourceBundle.getBundle("LabelsBundle", currentLocale);
```

5. lesen Sie den lokalen Text

```
String value = labels.getString(key);
```

6. [optional] lesen Sie die weiteren Werte, je einen pro Schlüssel

```
ResourceBundle labels =
    ResourceBundle.getBundle("LabelsBundle", currentLocale);
Enumeration bundleKeys = labels.getKeys();

while (bundleKeys.hasMoreElements()) {
    String key = (String)bundleKeys.nextElement();
    String value = labels.getString(key);
    System.out.println("Schlüssel = " + key + ", " +
                  "Wert = " + value);
}
```

1.3.1.7. Einsatz des ListResourceBundle

In diesem Teilabschnitt schauen wir uns an, wie man mit ListResourceBundle Klassen arbeiten kann. Das Beispielprogramm verwendet mehrere spezielle Klassen, in denen statistische Werte über die Bevölkerung einiger Länder stehen.

Das Beispiel zeigt neben dem Mechanischen (Art und Weise, wie man ListResourceBundle Klassen definiert) auch wie wichtig es ist, die Dateien im korrekten Verzeichnis abzuspeichern. Sie dürfen die Stats... Dateien nicht ins Package hineinnehmen, da sonst ein Problem beim Lesen und Laden der Dateien durch die util Klassen geschieht.

Nun zum Vorgehen:

1. kreieren der Unterklassen des ListResourceBundle
Das ListResourceBundle kennt in unserem Fall drei Locale Werte, je einen für en_CA, fr_FR und ja_JP. Pro Locale müssen wir eine Datei, eine Klasse zur Verfügung stellen

```
StatsBundle_en_CA.class
StatsBundle_fr_FR.class
StatsBundle_ja_JP.class
```

Der Name der Klassen ist gegeben aufgrund der Locals. In der Klasse wird einfach ein zweidimensionales Datenfeld definiert, welches Schlüssel, Wert Paare enthält.
In unserem Beispiel sind die Schlüssel: GDP, Population und Literacy.

Schlüssel kann nur eine Zeichenkette sein.
Die Schlüssel müssen in allen Dateien identisch sein.
Die Werte können beliebige Objekte sein.

```
// kein Package
import java.util.*;
public class StatsBundle_ja_JP extends ListResourceBundle {
    public Object[][] getContents() {
          return contents;
    }
    private Object[][] contents = {
          { "GDP", new Integer(21300) },
          { "Population", new Integer(125449703) },
          { "Literacy", new Double(0.99) },
    };
}
```

2. spezifizieren Sie die Locale
Im Demo Programm (s.unten) finden Sie die Angaben zu den Locals

```
Locale[] supportedLocales = {
    new Locale("en", "CA"),
    new Locale("ja", "JP"),
    new Locale("fr", "FR")
};
```

Jedes Locale entspricht einer Klasendatei.

3. Kreieren Sie das ResourceBundle

Ein ListResourceBundle wird mit der getBundle() Methode festgelegt:

```
ResourceBundle stats =
          ResourceBundle.getBundle("StatsBundle",
currentLocale);
```

Die getBundle() Methode sucht die Klassendatei (dort wo Sie bereits Ihre Properties Dateien hingeschrieben haben, also nicht im package Verzeichnis).

4. Fetch des Localized Objekts

Nun können wir die Daten aus den Klassen lesen:

```
Double lit = (Double)stats.getObject("Literacy");
```

Hier das vollständige Listing:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
public class ListResourceBundleBeispiel {
   static void displayValues(Locale currentLocale) {
      ResourceBundle stats =
         ResourceBundle.getBundle("StatsBundle",currentLocale);
      Integer gdp = (Integer)stats.getObject("GDP");
      System.out.println("GDP = " + gdp.toString());
      Integer pop = (Integer)stats.getObject("Population");
      System.out.println("Population = " + pop.toString());
      Double lit = (Double)stats.getObject("Literacy");
      System.out.println("Lese- und Schreibfähigkeit [Analphabetismusindex]
                  = " + lit.toString());
 } // displayValues

   static public void main(String[] args) {
      Locale[] supportedLocales = {
         new Locale("en","CA"),
         new Locale("ja","JP"),
         new Locale("fr","FR")
      };
      for (int i = 0; i < supportedLocales.length; i ++) {
         System.out.println("Locale = " + supportedLocales[i]);
         displayValues(supportedLocales[i]);
         System.out.println();
      }
   } // main
} // class
```

Und nun schauen wir uns die Ausgabe des Beispielprogramms an:

```
Locale = en_CA
Bruttosozialprodukt = 24400
Bevölkerung = 28802671
Lese- und Schreibfähigkeit [Analphabetismusindex] = 0.97

Locale = ja_JP
Bruttosozialprodukt = 21300
Bevölkerung = 125449703
Lese- und Schreibfähigkeit [Analphabetismusindex] = 0.99

Locale = fr_FR
Bruttosozialprodukt = 20200
Bevölkerung = 58317450
Lese- und Schreibfähigkeit [Analphabetismusindex] = 0.99
```

Behaften Sie mich nicht auf die Zahlen. Diese sind sicher bereits überholt.

1.3.2. Die ResourceBundle Klasse

Wir haben nun einige Beispiele für den praktischen Einsatz der Klasse ResourceBundle gesehen. Jetzt wollen wir noch die formalen Aspekte anschauen:

Jede Klasse, die aus der abstrakten Klasse ResourceBundle hergeleitet wird, definiert folgende Methoden:

`public final String getString(String key) throws MissingResourceException`
: liefert eine Zeichenkette, die im ResourceBundle unter dem Schlüssel key gespeichert wird.

`public final String[] getStringArray(String key)`
: liefert das String Array,welches unter dem Schlüssel key im resourceBundle gespeichert wird.

`public final Object getObject(String key) throws MissingResourceException`
: liefert das Objekt in ResourceBundle, welches zum Schlüssel key abgespeichert wird

`public abstract Enumeration getKeys()`
: liefert eine Auszählung der Schlüssel im ResourceBundle, alle Schlüssel.

Sie können selber Unterklassen der abstrakten Klasse ResourceBundle definieren. In der Regel werden aber ListResourceBundle, PropertyResourceBundle und die .properties Dateien ausreichen. Falls Sie eine eigene Klasse definieren möchten, müssen Sie folgende Methoden implementieren:

```
protected abstract Object handleGetObject(String key) throws
MissingResourceException
```
liefert das Objekt zum Schlüssel key. Falls zum Schlüssel kein Objekt definiert ist, wird ein leeres Objekt zurückgegeben und überprüft, ob in der Oberklasse ein Objekt zu diesem Schlüssel passt.

```
public abstract Enumeration getKeys()
```
liefert eine Aufzählung der Schlüssel, präziser : ein Enumeration Objekt, welches über die Schlüssel in diesem Bundle iteriert.

1.4. Formattierung

Nun geht es um die Details der Formattierung. Wir wollen anschauen,wie man Zahlen, Währung, Datum, Uhrzeit und Textmeldungen international korrekt formattieren kann.

1.4.1. Zahlen und Währung

Programme speichern die Daten in einer locale-unabhängigen Darstellung. Sie müssen also vor der Ausgabe, auf dem Bildschirm oder dem Drucker, die Daten passend (locale-gerecht) formattieren. Bei den Daten geht es um Hochkommata, Abstände bei Tausendern und vieles weitere mehr.

1.4.1.1. Der Einsatz vordefinierter Formate

Sie können mit Hilfe der Klasse `NumberFormat` Klasse können Sie numerische Daten, Währungen und Prozentdarstellungen gemäss den Locale Objekten formattieren.

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;


public class NumberFormatBeispiel {
   static public void displayNumber(Locale currentLocale) {

      Integer quantity = new Integer(123456);
      Double amount = new Double(345987.246);
      NumberFormat numberFormatter;
      String quantityOut;
      String amountOut;

      numberFormatter = NumberFormat.getNumberInstance(currentLocale);
      quantityOut = numberFormatter.format(quantity);
      amountOut = numberFormatter.format(amount);
      System.out.println("[Quantität]\t "+quantityOut);
      System.out.println("[Betrag]\t "+amountOut);
   }

   static public void displayCurrency(Locale currentLocale) {

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

      currencyFormatter = NumberFormat.getCurrencyInstance(currentLocale);
      currencyOut = currencyFormatter.format(currency);
      System.out.println("[Währung]\t "+currencyOut);
   }
```

```
   static public void displayPercent(Locale currentLocale) {

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

      percentFormatter = NumberFormat.getPercentInstance(currentLocale);
      percentOut = percentFormatter.format(percent);
      System.out.println("[Prozent]\t "+ percentOut );
   }

   static public void main(String[] args) {

      Locale[] locales = {
          new Locale("fr","FR"),
          new Locale("de","DE"),
          new Locale("en","US")
      };

      for (int i = 0; i < locales.length; i++) {
         System.out.println("\n"+locales[i].getDisplayCountry()+"\n");
         displayNumber(locales[i]);
         displayCurrency(locales[i]);
         displayPercent(locales[i]);
      }
   }

}
```

mit der Ausgabe

```
France

[Quantität]  123 456
[Betrag]     345 987,246
[Währung]    9 876 543,21 F
[Prozent]    75%

Deutschland

[Quantität]  123.456
[Betrag]     345.987,246
[Währung]    9.876.543,21 DM
[Prozent]    75%

Vereinigte Staaten

[Quantität]  123,456
[Betrag]     345,987.246
[Währung]    $9,876,543.21
[Prozent]    75%
```

Auch hier sehen Sie eine Inkonsistenz: bei Frankreich wird das Land in französisch, bei den USA in deutsch ausgegeben.

1.4.1.1.1. Numerische Objekte

Mit den Methoden der NumberFormat Methoden können Sie die Basisdatentypen formattieren, beispielsweise Doublezahlen und vieles mehr.

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

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

1.4.1.1.2. Währungen

Auch hier kreieren Sie eine Instanz der NumberFormat Klasse und verwenden deren getCurrencyInstance() Methode, um ein Währungsformattierobjekt zu kreieren.

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

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

1.4.1.1.3. Prozente

Falls Sie die Zahl 0.75 als Prozent auffassen, könnten Sie auch als 75% darstellen.

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

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

1.4.2. Anpassen der Formate

Sie können die Formattierung selber auch noch steuern. Dazu stehen Ihnen verschiedene Möglichkeiten zur Verfügung, die wir zuerst in einem einfachen Beipiel zusammenfassen.

1.4.2.1. Ein Beispiel

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class DezimalFormattierung {
   static public void customFormat(String pattern, double value ) {
      DecimalFormat meinFormattierer = new DecimalFormat(pattern);
      String output = meinFormattierer.format(value);
      System.out.println(value + "\t  " + pattern + "\t  " + output);
   }

   static public void localizedFormat(String pattern, double value,
                                      Locale loc ) {
      NumberFormat nf = NumberFormat.getNumberInstance(loc);
      DecimalFormat df = (DecimalFormat)nf;
      df.applyPattern(pattern);
      String output = df.format(value);
      System.out.println("["+loc.toString()+"]\t"+pattern + "\t  " +
output);
   }

   static public void main(String[] args) {

      customFormat("###,###.###", 123456.789);
      customFormat("###.##", 123456.789);
      customFormat("000000.000", 123.78);
      customFormat("$###,###.###", 12345.67);
      customFormat("\u00a5###,###.###", 12345.67);
      System.out.println();

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

      DecimalFormatSymbols unusualSymbols =
         new DecimalFormatSymbols(currentLocale);
      unusualSymbols.setDecimalSeparator('|');
      unusualSymbols.setGroupingSeparator('^');
      String strange = "#,##0.###";
      DecimalFormat weirdFormatter = new DecimalFormat(strange,
unusualSymbols);
      weirdFormatter.setGroupingSize(4);
      String bizarre = weirdFormatter.format(12345.678);
      System.out.println("[Exote]\t "+bizarre);
```

```
      Locale[] locales = {
         new Locale("en", "US"),
         new Locale("de", "DE"),
         new Locale("fr", "FR")
      };

      for (int i = 0; i < locales.length; i++) {
         localizedFormat("###,###.###", 123456.789, locales[i]);
      }

   }
}
```

Dieses liefert folgende Ausgabe:

```
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

[Exote]      1^2345|678
[en_US]     ###,###.###   123,456.789
[de_DE]     ###,###.###   123.456,789
[fr_FR]     ###,###.###   123 456,789
```

1.4.2.2. Formattiermuster

Die Formattierung geschieht mit Hilfe einer Zeichenkette. Die formale Syntax sehen Sie in folgendem BNF Diagramm:

```
pattern    := subpattern{;subpattern}
subpattern := {prefix}integer{.fraction}{suffix}
prefix     := '\\u0000'..'\\uFFFD' - specialCharacters
suffix     := '\\u0000'..'\\uFFFD' - specialCharacters
integer    := '#'* '0'* '0'
fraction   := '0'* '#'*
```

mit folgender Notation (Standard):

Notation	Beschreibung
X*	0 oder mehrere Instanzen von X
(X | Y)	entweder X oder Y
X..Y	ein Zeichen zwischen X und Y, inklusive
S - T	Zeichen in S, aber nicht in T
{X}	X ist optional

Symbol	Beschreibung
0	eine Zahl
#	eine Zahl ohne 0
.	Dezimalpunkt
,	Gruppierung
E	Exponentiale Darstellung
;	Formataufteilung
-	negativer Präfix
%	mal 100 und %
?	mal 1000 und Promille
¤	Währungssymbol
X	irgend einZeichen
'	Zeichen in Anführungszeichen

Beispiel:

```
DecimalFormat meinFormattierer = new DecimalFormat(pattern);
String output = meinFormattierer.format(value);
System.out.println(value + " " + pattern + " " + output);
```

Die folgende Tabelle zeigt die Formattierungsmuster, die im Demo Beispiel verwendet wurden.

Wert	Muster	Ausgabe	Beschreibung
123456.789	###,###.###	123,456.789	siehe oben
123456.789	###.##	123456.79	Formattierung mit Runden.
123.78	000000.000	000123.780	Formattierung mit führenden Nullen.
12345.67	$###,###.###	$12,345.67	führendes } Zeichen Wichtig: ohne Abstand
12345.67	\u00A5###,###.###	¥12,345.67	Japanischer Yen (¥) als Unicode 00A5.

1.4.2.3. Locale-Sensitive Formattierung

Oben haben wir die Standardlocals verwendet. Natürlich können Sie auch noch im Programm definierte Locals verwenden:

```
NumberFormat nf = NumberFormat.getNumberInstance(loc);

// ab hier wird local formattiert
DecimalFormat df = (DecimalFormat)nf;
df.applyPattern(pattern);
String output = df.format(value);
System.out.println(pattern+" "+output+" "+loc.toString());
```

1.4.2.4. Eigene Formattierungs Symbole

Mit Hilfe der DecimalFormatSymbols Klasse werden die Symbole in der formattierten Ausgabe angepasst:

die Methoden `setDecimalSeparator, setGroupingSeparator` und `setGroupingSize` Methoden.

```
DecimalFormatSymbols unusualSymbols =
    new DecimalFormatSymbols(currentLocale);
unusualSymbols.setDecimalSeparator('|');
unusualSymbols.setGroupingSeparator('^');

String strange = "#,##0.###";
DecimalFormat weirdFormatter =
               new DecimalFormat(strange, unusualSymbols);
weirdFormatter.setGroupingSize(4);

String bizarre = weirdFormatter.format(12345.678);
System.out.println(bizarre);
```

Damit erhalten Sie die folgende exotische Ausgabe:

```
1^2345|678
```

1.4.3. Datum und Uhrzeit

Date Objekte repräsentieren die Zeit und das Datum. Sie können kein Date Objekt ohne irgend eine Konvertierung durchzuführen. Die Formattierung des Datums ist sehr stark local-sensitive. Beispiel: der 20.4.98 in Deutschland ist gleich dem 4/20/98 in den USA.

Die `DateFormat` Klasse liefert Ihnen einige Methoden, mit denen Sie Ihr Date Objekt Ihren lokalen Anforderungen anpassen können.

1.4.3.1. Der Einsatz vordefinierter Formate

Schauen wir uns zuerst ein Beispiel für den Einsatz den `DateFormat` Klasse an:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class DateFormatBeispiel {
   static public void displayDate(Locale currentLocale) {
      Date today;
      String dateOut;
      DateFormat dateFormatter;
      dateFormatter =
         DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
      today = new Date();
      dateOut = dateFormatter.format(today);
      System.out.println("["+currentLocale.getDisplayCountry()+"]\t\t"+
dateOut );
   }

   static public void showBothStyles(Locale currentLocale) {
      Date today;
      String result;
      DateFormat formatter;
      int[] styles = {
         DateFormat.DEFAULT,
         DateFormat.SHORT,
         DateFormat.MEDIUM,
         DateFormat.LONG,
         DateFormat.FULL
      };

      System.out.println();
      System.out.println("Locale: " + currentLocale.toString()+"
["+currentLocale.getDisplayCountry()+"]");
      System.out.println();

      today = new Date();

      for (int k = 0; k < styles.length; k++) {
```

```
         formatter = DateFormat.getDateTimeInstance(
                     styles[k], styles[k], currentLocale);
         result = formatter.format(today);
         System.out.println(result);
      }
   }

   static public void showDateStyles(Locale currentLocale) {

      Date today = new Date();
      String result;
      DateFormat formatter;

      int[] styles = {
         DateFormat.DEFAULT,
         DateFormat.SHORT,
         DateFormat.MEDIUM,
         DateFormat.LONG,
         DateFormat.FULL
      };

      System.out.println();
      System.out.println("Locale: " + currentLocale.toString()+"
["+currentLocale.getDisplayCountry()+"]");
      System.out.println();

      for (int k = 0; k < styles.length; k++) {
         formatter =
            DateFormat.getDateInstance(styles[k], currentLocale);
         result = formatter.format(today);
         System.out.println(result);
      }
   }

   static public void showTimeStyles(Locale currentLocale) {

      Date today = new Date();
      String result;
      DateFormat formatter;

      int[] styles = {
         DateFormat.DEFAULT,
         DateFormat.SHORT,
         DateFormat.MEDIUM,
         DateFormat.LONG,
         DateFormat.FULL
      };

      System.out.println();
      System.out.println("Locale: " + currentLocale.toString()+"
["+currentLocale.getDisplayCountry()+"]");
      System.out.println();

      for (int k = 0; k < styles.length; k++) {
         formatter =
            DateFormat.getTimeInstance(styles[k], currentLocale);
         result = formatter.format(today);
         System.out.println(result);
      }
   }
```

```
   static public void main(String[] args) {

      Locale[] locales = {
          new Locale("fr","FR"),
          new Locale("de","DE"),
          new Locale("en","US")
      };

      for (int i = 0; i < locales.length; i++) {
         displayDate(locales[i]);
      }

      showDateStyles(new Locale("en","US"));
      showDateStyles(new Locale("fr","FR"));

      showTimeStyles(new Locale("en","US"));
      showTimeStyles(new Locale("de","DE"));

      showBothStyles(new Locale("en","US"));
      showBothStyles(new Locale("fr","FR"));

   }
}
```

Dieses produziert folgende Ausgabe:

```
[France]               5 mai 01
[Deutschland]          05.05.2001
[Vereinigte Staaten]   May 5, 2001

Locale: en_US  [Vereinigte Staaten]

May 5, 2001
5/5/01
May 5, 2001
May 5, 2001
Saturday, May 5, 2001

Locale: fr_FR  [France]

5 mai 01
05/05/01
5 mai 01
5 mai 2001
samedi 5 mai 2001

Locale: en_US  [Vereinigte Staaten]

4:54:37 PM
4:54    PM
4:54:37 PM
4:54:37 PM GMT+02:00
4:54:37 PM GMT+02:00

Locale: de_DE  [Deutschland]

16:54:37
16:54
16:54:37
16:54:37 GMT+02:00
```

```
16.54 Uhr GMT+02:00

Locale: en_US  [Vereinigte Staaten]

May 5, 2001 4:54:38 PM
5/5/01 4:54 PM
May 5, 2001 4:54:38 PM
May 5, 2001 4:54:38 PM GMT+02:00
Saturday, May 5, 2001 4:54:38 PM GMT+02:00

Locale: fr_FR  [France]

5 mai 01 16:54:38
05/05/01 16:54
5 mai 01 16:54:38
5 mai 2001 16:54:38 GMT+02:00
samedi 5 mai 2001 16 h 54 GMT+02:00
```

1.4.3.1.1. Datum

Die Formattierung des Datums geschieht mit der DateFormat Klasse in zwei Schritten.

1. Sie kreieren ein Formatobjekt mit Hilfe der getDateInstance() Methode.
2. Sie rufen die Formattiermethoden auf, welche eine Zeichenkettendarstellung für das formattierte Datum liefert.

Beispiel:

```
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());
```

Dieser Programmteil produziert folgende Ausgabe:

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

In diesem Beispiel verwenden wir die Standard (DateFormat.DEFAULT) Formatierung. Sie haben folgende fünf Möglichkeiten:

- DEFAULT
- SHORT
- MEDIUM
- LONG
- FULL

Beispiele für die unterschiedlichen Ausgaben sehen Sie im ersten Beispiel oben ganz am Ende.

1.4.3.1.2. Uhrzeit

Analog zum Datum erhalten Sie aus einem Date Objekt auch die Uhrzeit. Der Unterschied besteht in der Methode: `getTimeInstance()` liefert Ihnen die Uhrzeit

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

Beispielausgaben sehen Sie oben im ersten Beispiel.

Falls Sie die Uhrzeit und das Datum benötigen, könnten Sie beispielsweise folgendes Konstrukt anwenden:

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

1.4.3.2. Anpassen der Formattierung

In der Regel werden Sie mit einem der Standardformate auskommen. Aber Sie können auch eigene Formattierungen vornehmen, analog zur Formattierung bei den numerischen Ausgaben.

Hier zuerst ein Beispiel:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class EigeneDateFormate {

   static public void displayDate(Locale currentLocale) {

      Date today;
      String result;
      SimpleDateFormat formatter;

      formatter = new SimpleDateFormat("EEE d MMM yy", currentLocale);
      today = new Date();
      result = formatter.format(today);

      System.out.println("Locale: " + currentLocale.toString()+"
["+currentLocale.getDisplayLanguage()+"]");
      System.out.println("Resultat: " + result);
   }
```

```
   static public void displayPattern(String pattern, Locale currentLocale)
{

      Date today;
      SimpleDateFormat formatter;
      String output;

      formatter = new SimpleDateFormat(pattern, currentLocale);
      today = new Date();
      output = formatter.format(today);

      System.out.println("["+pattern+"]\t" + "   " + output);
   }

   static public void main(String[] args) {

      Locale[] locales = {
          new Locale("fr","FR"),
          new Locale("de","DE"),
          new Locale("en","US")
      };

      for (int i = 0; i < locales.length; i++) {
         displayDate(locales[i]);
         System.out.println();
      }

      String[] patterns = {
         "dd.MM.yy",
         "yyyy.MM.dd G 'at' hh:mm:ss z",
         "EEE, MMM d, ''yy",
         "h:mm a",
         "H:mm",
         "H:mm:ss:SSS",
         "K:mm a,z",
         "yyyy.MMMMM.dd GGG hh:mm aaa"
      };

      for (int k = 0; k < patterns.length; k++) {
         displayPattern(patterns[k], new Locale("en","US"));
      }

      System.out.println();
   }
}
```

mit folgender Ausgabe:

```
Locale: fr_FR [French]
Resultat: sam. 5 mai 01

Locale: de_DE [Deutsch]
Resultat: Sa 5 Mai 01

Locale: en_US [Englisch]
Resultat: Sat 5 May 01

[dd.MM.yy]     05.05.01
[yyyy.MM.dd G 'at' hh:mm:ss z]         2001.05.05 AD at 05:11:59 GMT+02:00
[EEE, MMM d, ''yy]         Sat, May 5, '01
```

```
[h:mm a]       5:11 PM
[H:mm]         17:11
[H:mm:ss:SSS]        17:11:59:809
[K:mm a,z]     5:11 PM,GMT+02:00
[yyyy.MMMMM.dd GGG hh:mm aaa]    2001.May.05 AD 05:12 PM
```

1.4.3.2.1. Formattiermuster

Die Klasse SimpleDateFormat hilft Ihnen dabei, eigene Ausgabemuster, in einem klar definierten Rahmen, zu generieren. Zur Ausgabe verwenden wir folgenden Programmcode:

```
Date today;
String output;
SimpleDateFormat formatter;

formatter = new SimpleDateFormat(pattern, currentLocale);
today = new Date();
output = formatter.format(today);
System.out.println(pattern + " " + output);
```

Mögliche Ausgaben sehen Sie oben.

1.4.3.3. Muster und Locale

Die `SimpleDateFormat` Klasse ist locale-sensitiv. Falls Sie ein Objekt der `SimpleDateFormat` Klasse instanzieren und kein Locale angeben, dann wird gemäss dem Standard-Locale formattiert.

Falls Sie dagegen ein Locale angeben, dann können Sie zusammen mit einem Formatmuster flexiblere Ausgabeformate generieren.

Hier ein Beispiel:

```
Date today;
String result;
SimpleDateFormat formatter;

formatter = new SimpleDateFormat("EEE d MMM yy",
                            currentLocale);
today = new Date();
result = formatter.format(today);
System.out.println("Locale  : " + currentLocale.toString());
System.out.println("Resultat: " + result);
```

Je nach dem Wert von currentLocale erhalten Sie eine der folgenden Ausgaben:

```
Locale  : fr_FR
Resultat: ven 10 avr 98
Locale  : de_DE
Resultat: Fr 10 Apr 98
Locale  : en_US
Resultat: Thu 9 Apr 98
```

1.4.3.4. Anpassen der Formatsymbole

Die Formatmethoden gestatten Ihnen die Formattierung wie im ersten Beispiel:

Methode zum Setzen	Beispiel
setAmPmStrings	PM
setEras	AD
setMonths	December
setShortMonths	Dec
setShortWeekdays	Tue
setWeekdays	Tuesday
setZoneStrings	PST

Im folgenden Beispiel wollen wir die Wochentage international ausgeben:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class DateFormatAnpassungen {
   static public void changeWeekDays() {

      Date today;
      String result;
      SimpleDateFormat formatter;
      DateFormatSymbols symbols, moreSymbols;
      String[] defaultDays;
      String[] modifiedDays;
      String[] moreDays;

      symbols = new DateFormatSymbols(new Locale("en","US"));
      moreSymbols = new DateFormatSymbols(new Locale("de","DE"));
      defaultDays = symbols.getShortWeekdays();
      moreDays = moreSymbols.getShortWeekdays();

      for (int i = 0; i < defaultDays.length; i++) {
         System.out.print(defaultDays[i] + "  ");
      }
      System.out.println();

      String[] capitalDays = {
         "", "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};
      symbols.setShortWeekdays(capitalDays);

      modifiedDays = symbols.getShortWeekdays();
      for (int i = 0; i < modifiedDays.length; i++) {
         System.out.print(modifiedDays[i] + "  ");
      }
      System.out.println();
```

```
      for (int i=0; i< moreDays.length; i++)
        System.out.print(moreDays[i]+"  ");
      System.out.println();

      formatter = new SimpleDateFormat("E", symbols);
      today = new Date();
      result = formatter.format(today);
      System.out.println(result);
   }

   static public void main(String[] args) {
      changeWeekDays();
   }
}
```

Dieses Beispiel generiert folgende Ausgabe:

```
Sun  Mon  Tue  Wed  Thu  Fri  Sat
SUN  MON  TUE  WED  THU  FRI  SAT
So  Mo  Di  Mi  Do  Fr  Sa
SAT
```

1.4.4. Kalender

Kalender betreffen die Änderungen der Zeit, deren Festschreibung. In den meisten Gegenden der Welt setzt man den selben Kalender ein, den Gregorianischen, nach Papst Gregor XIII. Aber es existieren viele weitere Kalenderarten. Daher hat man in Java einen abstrakten Kalender definiert. Die Kalender Abstraktion besteht aus:

- einer abstrakten Calendar Klasse, mit deren Hilfe unterschiedliche Zeitmarken definiert werden können.
- eine abstrakte Zeitzone mit der TimeZone Klasse, welche unter anderem die Sommerzeit berücksichtigt
- eine abstrakte java.text.DateFormat Klasse, mit deren Hilfe die Formate für Datum und Zeit festgelegt werden können.

Für den Gregorianischen Kalender wurden folgende Hilfsklassen definiert:

- eine GregorianCalendar Klasse.
- eine SimpleTimeZone Klasse, welche zusammen mit dem GregorianCalendar benutzt werden kann.
- eine java.text.SimpleDateFormat Klasse, mit deren Hilfe die Objekte des GregorianCalendar formattiert und zerlegt werden kann.

Beispiel:

```
Calendar cal = new GregorianCalendar(1972, Calendar.OCTOBER, 26);
System.out.println(cal.getTime() );
```

Ergebnis:

```
Thu Oct 26 00:00:00 GMT+02:00 1972
```

wobei das Programm in der Schweiz ausgeführt wurde.

1.4.5. Messages

Bei komplexeren Programmen ist es wichtig, dass Sie gelegentlich eine Meldung ausgeben oder eventuell einen Progressbar einbauen. Diese Meldungen müssen übersetzt werden und auf der Statuszeile angezeigt werden. Ganze Meldungszeilen werden Sie kaum in ein ResourceBundle verschieben, weil die Meldung berechnete Daten enthält. In diesem Fall verbietet Ihnen der unterschiedliche Satzaufbau in den unterschiedlichen Sprachen ein Auslagern in ein ResourceBundle.

Schauen wir uns eine etwas komplexere Meldung an:

```
The disk named MyDisk contains 300 files.
The current balance of account #34-98-222 is $2,745.72.
405,390 people have visited your website since January 1, 1998.
Delete all files older than 120 days.
```

Nun können wir als erstes versuchen, diese Meldung in ein EsourceBundle zu verschieben:
double numDays;

```
ResourceBundle msgBundle;
...
String message = msgBundle.getString("deleteolder"
                                     + numDays.toString()
                                     + msgBundle.getString("days"));
```

Das funktioniert im Englischen, aber nicht mit dem deutschen Text. Aber wir können das Format in einem ResourceBundle speichern.

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */

import java.util.*;
import java.text.*;

public class MessageFormatBeispiel {

   static  void displayMessage(Locale currentLocale) {

      System.out.println("aktuelles Locale Objekt= " +
currentLocale.toString()+"\t["+currentLocale.getDisplayCountry()+"]");

      ResourceBundle messages =
         ResourceBundle.getBundle("MessageBundle2",currentLocale);

      Object[] messageArguments = {
         messages.getString("planet"),
         new Integer(7),
         new Date()
      };
```

```
      MessageFormat formatter = new MessageFormat("");
      formatter.setLocale(currentLocale);

      formatter.applyPattern(messages.getString("template"));
      String output = formatter.format(messageArguments);

      System.out.println(output);

   }

   static public void main(String[] args) {
      displayMessage(new Locale("en", "US"));
      System.out.println();
      displayMessage(new Locale("de", "DE"));
   }
}
```

mit der Ausgabe:

```
aktuelles Locale Objekt= en_US     [Vereinigte Staaten]
At 7:00 PM on May 5, 2001, we detected 7 spaceships on the planet Mars.

aktuelles Locale Objekt= de_DE     [Deutschland]
Um 19:00 am 5. Mai 2001 haben wir 7 Raumschiffe auf dem Planeten Mars
entdeckt.
```

mit dem ResourceBundle:

```
# MessageBundle.properties
planet = Mars
color = purpurne
template = Um {2,time,short} am {2,date,long} haben wir {1,number,integer}
Raumschiffe auf dem Planeten {0} entdeckt.
```

Un nun das schrittweise Vorgehen:

1. Identifizieren Sie die Variablen in den Meldungen
   Im obigen Text sind diese klar erkennbar (Uhrzeit, Datum, Anzahl).

2. Isolieren Sie das Message Pattern in einem ResourceBundle
   Speichern Sie die Meldung in einem ResourceBundle, hier "MessageBundle2" (da wir MessageBundle in einführenden Beispiel schon verbraucht haben).

   ```
   ResourceBundle messages =
       ResourceBundle.getBundle("MessageBundle", currentLocale);
   ```
   Den Inhalt der Datei sehen Sie oben.

3. Im ResourceBundle beschreiben Sie das Muster der Antwort. Als Platzhalter sehen Sie die Variablen in '{' Klammern. Das erste Zeichen ist eine Zahl, der Index des Objekts im Array der Argumentwerte.
   In unserem Fall wird zuerst der Name des Planeten bestimmt.
   Die zweite Zeile wird nicht benötigt.
   In der dritten Zeile wird der Text vorgegeben.
   Im messageArguments Arrays wird zuerst der Planet (0), dass eine Integer(1) und schliesslich ein Date(2) Objekt aufgeführt. Die Formattierung steht gleich dahinter.

```
Object[] messageArguments = {
    messages.getString("planet"),
    new Integer(7),
    new Date()
};
```

1.4.5.1. Einzahl / Mehrzahl

Betrachten wir ein einfaches Beispiel:

```
There are no files on XDisk.
There is one file on XDisk.
There are 2 files on XDisk.
```

Dieser Text könnte in einem ersten Ansatz mit folgendem MessageFormat gelöst werden:

```
There are {0,number} file(s) on {1}.
```

Das führt aber zu einem Problem:

```
There are 1 file(s) on XDisk.
```

Das können Sie besser mit der `ChoiceFormat` Klasse lösen.

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class ChoiceFormatBeispiel {
   static void displayMessages(Locale currentLocale) {

      System.out.println("currentLocale = " + currentLocale.toString());
      System.out.println();

      ResourceBundle bundle =
         ResourceBundle.getBundle("ChoiceBundle",currentLocale);

      MessageFormat messageForm = new MessageFormat("");
      messageForm.setLocale(currentLocale);

      double[] fileLimits = {0,1,2};

      String [] fileStrings = {
         bundle.getString("noFiles"),
         bundle.getString("oneFile"),
         bundle.getString("multipleFiles")
      };

      ChoiceFormat choiceForm = new ChoiceFormat(fileLimits, fileStrings);

      String pattern = bundle.getString("pattern");
```

```
      Format[] formats = {choiceForm, null, NumberFormat.getInstance()};

      messageForm.applyPattern(pattern);
      messageForm.setFormats(formats);

      Object[] messageArguments = {null, "XDisk", null};

      for (int numFiles = 0; numFiles < 4; numFiles++) {
         messageArguments[0] = new Integer(numFiles);
         messageArguments[2] = new Integer(numFiles);
         String result = messageForm.format(messageArguments);
         System.out.println(result);
      }
   }

   static public void main(String[] args) {
      displayMessages(new Locale("en", "US"));
      System.out.println();
      displayMessages(new Locale("fr", "FR"));
      System.out.println();
      displayMessages(new Locale("de", "CH"));
   }
}
```

Schrittweises Vorgehen:

1. Definieren Sie das Message Pattern

```
      There {0} on {1}.
```

2. Kreieren Sie ein ResourceBundle

```
      ResourceBundle bundle =
         ResourceBundle.getBundle("ChoiceBundle", currentLocale);

      pattern = There {0} on {1}.
      noFiles = are no files
      oneFile = is one file
      multipleFiles = are {2} files


      pattern = Il {0} sur {1}.
      noFiles = n'y a pas de fichiers
      oneFile = y a un fichier

      # Choice.properties_de_CH
      noFiles = sind keine Dateien
      oneFile = ist eine Datei
      multipleFiles = sind {2} Dateien
      pattern = Es {0} auf {1}.
```

3. Kreieren eines Message Formatter

```
      MessageFormat messageForm = new MessageFormat("");
      messageForm.setLocale(currentLocale);
```

4. Kreieren eines Choice Formatter

```
double[] fileLimits = {0,1,2};
String [] fileStrings = {
    bundle.getString("noFiles"),
    bundle.getString("oneFile"),
    bundle.getString("multipleFiles")
};


ChoiceFormat choiceForm = new ChoiceFormat(fileLimits,
                                           fileStrings);
```

5. Anwenden des Patterns

```
String pattern = bundle.getString("pattern");
messageForm.applyPattern(pattern);
```

6. Zuweisen der Formate

```
Format[] formats = {choiceForm, null,
                    NumberFormat.getInstance()};
messageForm.setFormats(formats);
```

7. Zusammensetzen und formattieren

```
Object[] messageArguments = {null, "XDisk", null};
for (int numFiles = 0; numFiles < 4; numFiles++) {
    messageArguments[0] = new Integer(numFiles);
    messageArguments[2] = new Integer(numFiles);
    String result = messageForm.format(messageArguments);
    System.out.println(result);
}
```

Die Ausgabe :

```
currentLocale = en_US

There are no files on XDisk.
There is one file on XDisk.
There are 2 files on XDisk.
There are 3 files on XDisk.

currentLocale = fr_FR

Il n' y a pas des fichiers sur XDisk.
Il y a un fichier sur XDisk.
Il y a 2 fichiers sur XDisk.
Il y a 3 fichiers sur XDisk.

currentLocale = de_CH

Es sind keine Dateien auf XDisk.
Es ist eine Datei auf XDisk.
Es sind 2 Dateien auf XDisk.
Es sind 3 Dateien auf XDisk.
```

1.5. Arbeiten mit Text

Bei generellem Text in Ihren Programmen können Sie mit weiteren Hilfsklassen Ihre Programme für den internationalen Einsatz vorbereiten.

1.5.1. Überprüfen von Zeicheneigenschaften

Zur Einleitung ein Programmfragment, zuerst falsch, dann richtig geschrieben:

```
char ch;
...

// FEHLERHAFT!

if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
  // ch is a letter
...
if (ch >= '0' && ch <= '9')
  // ch is a digit
...
if ((ch == ' ') || (ch =='\n') || (ch == '\t'))
  // ch is a whitespace
```

Das Programmfragment ist nicht korrekt, weil Sie einen Zeichenvergleich durchführen. Java verwendet aber Unicode. Zudem stellt Ihnen Java jede Menge Methoden zur Verfügung, mit deren Hilfe das obige Programm korrekt formuliert werden kann:

```
char ch;
...

// Alles OKAY!

if (Character.isLetter(ch))
...
if (Character.isDigit(ch))
...
if (Character.isSpaceChar(ch))
```

Die Character Klasse stellt Ihnen verschiedene Prüfungsmethoden zur Verfügung, mit deren Hilfe Sie das Programm korrekt formulieren können. Das Programm arbeitet auch mit arabischen Zeichen korrekt.

Die Character Klasse liefert Ihnen folgende Methoden:

- `isDigit`
- `isLetter`
- `isLetterOrDigit`
- `isLowerCase`
- `isUpperCase`
- `isSpaceChar`
- `isDefined`

Auch einVergleich auf Gross- oder Kleinbuchstaben ist bereits vorgesehen:

```
if (Character.getType('a') == Character.LOWERCASE_LETTER)
...
if (Character.getType('R') == Character.UPPERCASE_LETTER)
...
if (Character.getType('>') == Character.MATH_SYMBOL)
...
if (Character.getType('_') == Character.CONNECTOR_PUNCTUATION)
```

1.5.2. Vergleich von Zeichenketten

Um internationale Zeichenketten miteinander einfach vergleichen zu können, wurde die Klasse `Collator` einegführt.

1.5.2.1. Performing Locale-Independent Comparisons

Mit der Collator Klasse sind Sie in der Lage internationale Zeichenketten miteinander zu vergleichen und beispielsweise zu sortieren.

Schauen wir uns wieder ein Beispiel an:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class CollatorBeispiel {
   public static void sortStrings(Collator collator, String[] words) {
       String tmp;
       for (int i = 0; i < words.length; i++) {
           for (int j = i + 1; j < words.length; j++) {
               // Elemente vergleichen
               if (collator.compare(words[i], words[j] ) > 0 ) {
                   // Swap words[i] und words[j]
                   tmp = words[i];
                   words[i] = words[j];
                   words[j] = tmp;
               }
           }
       }
   }

   public static void printStrings(String [] words) {
       for (int i = 0; i < words.length; i++) {
          System.out.println(words[i]);
       }
   }

   public static void testCompare() {
```

```
      Collator meinCollator = Collator.getInstance(new Locale("en", "US"));

      System.out.println("compare(\"abc\", \"def\")
"+meinCollator.compare("abc", "def")+" [\"abc\" < \"def\"]");
      System.out.println("compare(\"trf\", \"rtf\")
"+meinCollator.compare("rtf", "rtf")+" [\"rtf\" = \"rtf\"]");
      System.out.println("compare(\"xyz\", \"abc\")
"+meinCollator.compare("xyz", "abc")+" [\"xyz\" > \"abc\"]");
   }

   static public void main(String[] args) {

      testCompare();
      System.out.println();

      Collator fr_FRCollator = Collator.getInstance(new Locale("fr","FR"));
      Collator en_USCollator = Collator.getInstance(new Locale("en","US"));

      String eWithCircumflex = new String("\u00EA");
      String eWithAcute = new String("\u00E9");

      String peachfr = "p" + eWithAcute + "ch" + eWithAcute;
      String sinfr = "p" + eWithCircumflex + "che";

      String [] words = {
        peachfr,
        sinfr,
        "peach",
        "sin"
      };

      sortStrings(fr_FRCollator, words);
      System.out.println("Locale: fr_FR");
      printStrings(words);

      System.out.println();

      sortStrings(en_USCollator, words);
      System.out.println("Locale: en_US");
      printStrings(words);
   }
}
```

mit der Ausgabe:

```
compare("abc", "def")  -1 ["abc" < "def"]
compare("trf", "rtf")  0 ["rtf" = "rtf"]
compare("xyz", "abc")  1 ["xyz" > "abc"]

Locale: fr_FR
peach
pêche
péché
sin

Locale: en_US
peach
péché
pêche
sin
```

1.5.2.2. Anpassen der Collation Rules

Im obigen Beispiel haben wir die vordefinierten Regeln benutzt. Diese können Sie auch überchreiben, wobei man sich dann fragen muss, ob der Aufwand gerechtfertigt ist, speziell wenn Sie die Übung für 20 Sprachen machen wollen:

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class RuleBasedCollatorBeispiel {
   public static void sortStrings(Collator collator, String[] words) {
       String tmp;
       for (int i = 0; i < words.length; i++) {
           for (int j = i + 1; j < words.length; j++) {
               // Compare elements of the words array
               if( collator.compare(words[i], words[j] ) > 0 ) {
                   // Swap words[i] and words[j]
                   tmp = words[i];
                   words[i] = words[j];
                   words[j] = tmp;
               }
           }
       }
   }

   public static void printStrings(String [] words) {
       for (int i = 0; i < words.length; i++) {
          System.out.println(words[i]);
       }
   }

   static public void main(String[] args) {

      String englishRules =
         ("< a,A < b,B < c,C < d,D < e,E < f,F " +
          "< g,G < h,H < i,I < j,J < k,K < l,L " +
          "< m,M < n,N < o,O < p,P < q,Q < r,R " +
          "< s,S < t,T < u,U < v,V < w,W < x,X " +
          "< y,Y < z,Z");

      String smallnTilde  = new String("\u00F1");
      String capitalNTilde = new String("\u00D1");

      String traditionalSpanishRules =
         ("< a,A < b,B < c,C " +
          "< ch, cH, Ch, CH "  +
          "< d,D < e,E < f,F " +
          "< g,G < h,H < i,I < j,J < k,K < l,L " +
          "< ll, lL, Ll, LL "  +
```

```
            "< m,M < n,N " +
            "< " + smallnTilde + "," + capitalNTilde + " " +
            "< o,O < p,P < q,Q < r,R " +
            "< s,S < t,T < u,U < v,V < w,W < x,X " +
            "< y,Y < z,Z");

        String [] words = {
          "luz",
          "curioso",
          "llama",
          "chalina"
        };

        try {
            RuleBasedCollator enCollator =
               new RuleBasedCollator(englishRules);
            RuleBasedCollator spCollator =
               new RuleBasedCollator(traditionalSpanishRules);

           System.out.println("RuleBasedCollator : ");
           System.out.println("unsortierte Wörter");
           for (int i=0; i<words.length; i++)
             System.out.print(words[i]+" ");
           System.out.println();
           System.out.println();

           System.out.println("Englisch sortiert");
            sortStrings(enCollator, words);
            printStrings(words);

            System.out.println();

           System.out.println("Traditionell Spanisch sortiert");
            sortStrings(spCollator, words);
            printStrings(words);
        } catch (ParseException pe) {
            System.out.println("Parse Exception ");
        }
    }
}
```

mit der Ausgabe:

```
RuleBasedCollator :
unsortierte Wörter
luz curioso llama chalina

Englisch sortiert
chalina
curioso
llama
luz

Traditionell Spanisch sortiert
curioso
chalina
luz
llama
```

Sie müssen in diesem Beispiel die Regeln selber definieren:

```
String englishRules =
    ("< a,A < b,B < c,C < d,D < e,E < f,F " +
     "< g,G < h,H < i,I < j,J < k,K < l,L " +
     "< m,M < n,N < o,O < p,P < q,Q < r,R " +
     "< s,S < t,T < u,U < v,V < w,W < x,X " +
     "< y,Y < z,Z");

String smallnTilde = new String("\u00F1"); // ñ
String capitalNTilde = new String("\u00D1"); // Ñ

String traditionalSpanishRules =
    ("< a,A < b,B < c,C " +
     "< ch, cH, Ch, CH " +
     "< d,D < e,E < f,F " +
     "< g,G < h,H < i,I < j,J < k,K < l,L " +
     "< ll, lL, Ll, LL " +
     "< m,M < n,N " +
     "< " + smallnTilde + "," + capitalNTilde + " " +
     "< o,O < p,P < q,Q < r,R " +
     "< s,S < t,T < u,U < v,V < w,W < x,X " +
     "< y,Y < z,Z");
```

Die folgenden Zeilen instanzieren die Collatoren und sortieren die Wörter:

```
try {
    RuleBasedCollator enCollator =
        new RuleBasedCollator(englishRules);
    RuleBasedCollator spCollator =
        new RuleBasedCollator(traditionalSpanishRules);
    // ...
    sortStrings(enCollator, words);
    printStrings(words);
    //...
    sortStrings(spCollator, words);
    printStrings(words);
} catch (ParseException pe) {
    System.out.println("Parse Exception ");
}
```

Zum Sortieren benötigen wir den Collator-Vergleich:

```
public static void sortStrings(Collator collator, String[]
words) {
    String tmp;
    for (int i = 0; i < words.length; i++) {
  for (int j = i + 1; j < words.length; j++) {
      if (collator.compare(words[i], words[j]) > 0) {
          tmp = words[i];
          words[i] = words[j];
          words[j] = tmp;
      }
  }
 }
}
```

1.5.2.3. Verbessern der Collation Performance

Das Sortieren längerer Zeichenketten ist recht zeitaufwendig. Um die Performance zu steigern, wurde die Klasse `CollationKey` eingeführt.
Hier ein Beispielprogramm:

```
package einführendesbeispiel;
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;

public class CollatorKeyBeispiel {
   public static void sortArray(CollationKey[] keys) {
       CollationKey tmp;
       for (int i = 0; i < keys.length; i++) {
          for (int j = i + 1; j < keys.length; j++) {
             // Vergleich zweier Keys
             if( keys[i].compareTo( keys[j] ) > 0 ) {
                // Swap keys[i] und keys[j]
                tmp = keys[i];
                keys[i] = keys[j];
                keys[j] = tmp;
             }
          }
       }
   }
   static void displayWords(CollationKey[] keys) {
      for (int i = 0; i < keys.length; i++) {
                System.out.println(keys[i].getSourceString());
      }
   }
   static public void main(String[] args) {
      Collator enUSCollator = Collator.getInstance(new Locale("en","US"));
      String [] words = {
        "peach",        "apricot",        "grape",        "lemon"      };
      CollationKey[] keys = new CollationKey[words.length];
      for (int k = 0; k < keys.length; k ++) {
         keys[k] = enUSCollator.getCollationKey(words[k]);
      }

      sortArray(keys);
      displayWords(keys);
   }
}
```

Ausgabe:

```
      apricot
      grape
      lemon
      peach
```

1.5.3. Finden von Textgrenzen

Anwendungen, welche Text bearbeiten und darstellen, müssen die Grenzen / Bgrenzung einzelner Worte bestimmen können. Dafür stellt Java die `BreakIterator` Klasse zur Verfügung.

1.5.3.1. Die BreakIterator Klasse

Die BreakIterator Klasse ist locale-sensitive, weil die Grenzen eines Textes sprachabhängig sein können, speziell das Trennen von Wörtern. Welche Locales von einem BreakIterator unterstützt werden hängt von Ihrem Programm ab. Sie können die Locales jederzeit mit folgender Methode bestimmen:

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

Die BreakIterator Klasse ist entwickelt worden, um folgende Grenzen im Text zu bestimmen: Zeichen, Wörter, Sätze und mögliche Zeilenumbrüche. Dementsprechend gibt es die vier Methoden:

- getCharacterInstance
- getWordInstance
- getSentenceInstance
- getLineInstance

```
package einführendesbeispiel;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2000
 * @author J.M.Joller
 * @version 1.0
 */
import java.util.*;
import java.text.*;
import java.io.*;

public class BreakIteratorBeispiel {
   static void extractWords(String target, BreakIterator wordIterator) {

      wordIterator.setText(target);
      int start = wordIterator.first();
      int end = wordIterator.next();

      while (end != BreakIterator.DONE) {
         String word = target.substring(start,end);
         if (Character.isLetterOrDigit(word.charAt(0))) {
            System.out.println(word);
         }
         start = end;
         end = wordIterator.next();
      }
   }

   static void reverseWords(String target, BreakIterator wordIterator) {

      wordIterator.setText(target);
```

```
      int end = wordIterator.last();
      int start = wordIterator.previous();

      while (start != BreakIterator.DONE) {
         String word = target.substring(start,end);
         if (Character.isLetterOrDigit(word.charAt(0)))
            System.out.println(word);
         end = start;
         start = wordIterator.previous();
      }
   }

   static void markBoundaries(String target, BreakIterator iterator) {

      StringBuffer markers = new StringBuffer();
      markers.setLength(target.length() + 1);
      for (int k = 0; k < markers.length(); k++) {
         markers.setCharAt(k,' ');
      }

      iterator.setText(target);
      int boundary = iterator.first();

      while (boundary != BreakIterator.DONE) {
         markers.setCharAt(boundary,'^');
         boundary = iterator.next();
      }

      System.out.println(target);
      System.out.println(markers);
   }

   static void formatLines(String target, int maxLength,
                           Locale currentLocale) {

      BreakIterator boundary =
BreakIterator.getLineInstance(currentLocale);
      boundary.setText(target);
      int start = boundary.first();
      int end = boundary.next();
      int lineLength = 0;

      while (end != BreakIterator.DONE) {
         String word = target.substring(start,end);
         lineLength = lineLength + word.length();
         if (lineLength >= maxLength) {
            System.out.println();
            lineLength = word.length();
         }
         System.out.print(word);
         start = end;
         end = boundary.next();
      }
   }

   static void listPositions(String target, BreakIterator iterator) {

      iterator.setText(target);
      int boundary = iterator.first();

      while (boundary != BreakIterator.DONE) {
```

```
      System.out.println (boundary);
      boundary = iterator.next();
   }
}

static void characterExamples() {
   // arabischer CharacterIterator
   BreakIterator arCharIterator =
      BreakIterator.getCharacterInstance(new Locale ("ar","SA"));
   // arabische Wort für "Haus"
   String house = "\u0628" + "\u064e" + "\u064a" +
                  "\u0652" + "\u067a" + "\u064f";
   listPositions (house,arCharIterator);
}

static void wordExamples() {

   Locale currentLocale = new Locale ("en","US");
   BreakIterator wordIterator =
      BreakIterator.getWordInstance(currentLocale);
   String someText = "She stopped.  " +
                     "She said, \"Hello there,\" and then went on.";
   markBoundaries(someText, wordIterator);
   System.out.println();
   extractWords(someText, wordIterator);
}

static void sentenceExamples() {

   Locale currentLocale = new Locale ("en","US");
   BreakIterator sentenceIterator =
      BreakIterator.getSentenceInstance(currentLocale);
   String someText = "She stopped.  " +
                     "She said, \"Hello there,\" and then went on.";
   markBoundaries(someText, sentenceIterator);
   String variousText = "He's vanished!  " +
                        "What will we do?  It's up to us.";
   markBoundaries(variousText, sentenceIterator);
   String decimalText = "Please add 1.5 liters to the tank.";
   markBoundaries(decimalText, sentenceIterator);
   String  donneText = "\"No man is an island . . . " +
                       "every man . . . \"";
   markBoundaries(donneText, sentenceIterator);
   String dogText = "My friend, Mr. Jones, has a new dog.  " +
                    "The dog's name is Spot.";
   markBoundaries(dogText, sentenceIterator);
}

static void lineExamples() {

   Locale currentLocale = new Locale ("en","US");
   BreakIterator lineIterator =
     BreakIterator.getLineInstance(currentLocale);
   String someText = "She stopped.  " +
                     "She said, \"Hello there,\" and then went on.";
   markBoundaries(someText, lineIterator);
   String hardHyphen = "There are twenty-four hours in a day.";
   markBoundaries(hardHyphen, lineIterator);
   System.out.println();
   String moreText = "She said, \"Hello there,\" and then " +
                     "went on down the street.  When she stopped " +
```

```
                        "to look at the fur coats in a shop window, " +
                        "her dog growled.  \"Sorry Jake,\" she said. " +
                        " \"I didn't know you would take it personally.\"";
      formatLines(moreText, 30, currentLocale);
      System.out.println();
   }

   static public void main(String[] args) {

      characterExamples();
      System.out.println();
      wordExamples();
      System.out.println();
      sentenceExamples();
      System.out.println();
      lineExamples();
   }

} // class
```

Ausgabe:

```
0
2
4
6
She stopped.  She said, "Hello there," and then went on.
^  ^^      ^^ ^  ^^   ^^^^    ^^    ^^^^  ^^   ^^   ^^ ^^
She
stopped
She
said
Hello
there
and
then
went
on
She stopped.  She said, "Hello there," and then went on.
^             ^                                 ^
He's vanished!  What will we do?  It's up to us.
^              ^                ^             ^
Please add 1.5 liters to the tank.
^                                ^
"No man is an island . . . every man . . . "
^                      ^ ^             ^ ^ ^^
My friend, Mr. Jones, has a new dog.  The dog's name is Spot.
^             ^                      ^                       ^
She stopped.  She said, "Hello there," and then went on.
^   ^         ^   ^     ^      ^       ^   ^    ^    ^  ^
There are twenty-four hours in a day.
^     ^   ^      ^    ^     ^  ^ ^   ^
She said, "Hello there," and
then went on down the
street.  When she stopped to
look at the fur coats in a
shop window, her dog
growled.  "Sorry Jake," she
said.  "I didn't know you
would take it personally."
```