Lokalizacja i internacjonalizacja programów

Zaawansowane technologie Javy 2019

Lokalizacja

- Lokalizacja to specyficzne dla danego języka, kraju i regionu reguły dotyczące prezentacji różnych informacji (formatowania liczb, formatowania dat, pisowni tekstów, porządku alfabetycznego, itp).
- Programy powinny być dostosowane do różnych lokalizacji bez ponownego kompilowania kodu.

Lokalizacja

- W pakiecie `java.text` mamy cały zestaw klas rozwiązujący zagadnienia lokalizacyjne.
- Lokalizacja w Javie jest reprezentowana przez klasę `Locale` z pakietu `java.util`.
- Lokalizacja jest określana przez kombinację kodu języka, kraju i wariantu (regionu):
  - Kod języka to dwuliterowy skrót (małe litery) wg standardu ISO-639
  - Kod kraju to dwuliterowy skrót (duże litery) wg standardu ISO-3166
  - Kod wariantu (regionu) to dodatkowa informacja, która nie spełnia określonych standardów.

Lokalizacja

- Lokalizacja w Javie jest określana przez kombinację kodu języka, kraju i wariantu (regionu):
```
Locale (String lang)
Locale (String lang, String coun)
Locale (String lang, String coun, String var)
```
- Przykłady obiektów lokalizacyjnych:
```
Locale a = new Locale("pl","PL");
Locale b = new Locale("en","GB");
Locale c = new Locale("en","US");
Locale d = new Locale("en");
```
- Każdy program w Javie uzyskuje domyślną lokalizację na podstawie właściwości ustawionych dla platformy systemowej.
- Bieżącą lokalizację możemy odczytać za pomocą:
```
Locale.getDefault();
```
- Domyślną lokalizację można zmienić za pomocą:
```
Locale.setDefault(newLocale);
```

Lokalizacja

- Klasa `Locale` posiada kilka użytecznych getterów:
```
String getLanguage()
String getCountry()
String getVariant()
```
- Klasa `Locale` posiada kilka metod prezentacyjnych:
```
String getDisplayLanguage();
String getDisplayCountry();
String getDisplayVariant();
String getDisplayName();
```
- Klasa `Locale` posiada kilka informacyjnych metod statycznych:
```
String[] getISOLanguages()
String[] getISOCountries()
Locale[] getAvailableLocales()
```

Lokalizacja

- Klasy lokalizacyjnie czułe biorą pod uwagę lokalizację – zaliczamy do nich:
  `NumberFormat` (formatowanie liczb)
  `DateFormat` (formatowanie dat)
  `Calendar` (data i czas)
  `Collator` (porządek alfabetyczny)
  `BreakIterator` (rozbiór tekstu)

Formatowanie

- Do formatowania liczb (całkowitych i zmiennopozycyjnych) służą klasy `NumberFormat` i `DecimalFormat`.
- Klasa `Currency` opisuje waluty.
- Strefy czasowe opisuje klasa `TimeZone`.
- Informację o dacie i godzinie przechowuje się w obiekcie `Calendar` lub `Date`, a do formatowania czasu używa się klasy `DateFormat`.

Rozbiór tekstów

- Zlokalizowanego rozbioru tekstu dokonujemy za pomocą klasy `BreakIterator`.
- `BreakIterator` pozwala dzielić tekst pisany w określonym języku na wiersze, zdania, słowa i znaki. Do wyodrębnienia każdego z wymienionych elementów tekstu należy użyć odrębnego iteratora:
  `getLineInstance()`
  `getSentenceInstance()`
  `getWordInstance()`
  `getCharacterInstance()`
- Przykład:

```
BreakIterator brit =
  BreakIterator.getWordInstance();
```

Rozbiór tekstów

- Najważniejsze metody iteratorów:
  `first ()` – indeks pierwszej pozycji podziału tekstu na elementy;
  `last ()` – indeks ostatniej pozycji;
  `next ()` – indeks następnej pozycji;
  `previous ()` – indeks poprzedniej pozycji;
  `next (int n)` i `preceding (int n)` – indeks pozycji podziału tekstu oddalonego o `n` od pozycji bieżącej.
- Iterator do wyodrębniania słów, w przeciwieństwie do innych iteratorów, zwraca indeksy słów i indeksy separatorów.

Rozbiór tekstów

Przykład rozbioru tekstu:

```
public static void main (String args[]) {
  if (args.length == 1) {
    String stringToExamine = args[0];
    // print each word in order
    BreakIterator boundary =
      BreakIterator.getWordInstance();
    boundary.setText(stringToExamine);
    printEachForward(boundary,stringToExamine);
    // print each sentence in reverse order
    boundary = BreakIterator.getSentenceInstance(Locale.US);
    boundary.setText(stringToExamine);
    printEachBackward(boundary,stringToExamine);
    printFirst(boundary,stringToExamine);
    printLast(boundary,stringToExamine);
  }
}
```

Rozbiór tekstów

Przykład rozbioru tekstu:

```
public static void printEachForward
(BreakIterator boundary, String source) {
  int start = boundary.first();
  for (int end=boundary.next();
    end!=BreakIterator.DONE;
    start=end, end=boundary.next()) {
      System.out.println(source.substring(start,end));
  }
}
public static void printEachBackward
(BreakIterator boundary, String source) {
  int end = boundary.last();
  for (int start=boundary.previous();
    start!=BreakIterator.DONE;
    end=start, start=boundary.previous()) {
      System.out.println(source.substring(start,end));
  }
}
```

Porównywanie napisów

- Różne języki mają różny porządek alfabetyczny napisów.
- Właściwe porównywanie napisów dokonuje się za pomocą lokalizacyjnie czułego obiektu `Collator`.
- Klasa `Collator` implementuje interfejs `Comparator<Object>`.
- Klasa `Collator` jest abstrakcyjna.

Porównywanie napisów

- Konkretne kolatory są nieabstrakcyjnymi obiektami `RuleBasedCollator` (podklasa `Collator`).
- Instancję kolatora uzyskujemy za pomocą metod: `Collator.getInstance()` – kolator domyślny `Collator.getInstance(loc)` – kolator dla lokalizacji `loc`
- Metodą `compare(…)` kolatora porównujemy parę napisów.

Porównywanie napisów

- Uporządkowanie za pomocą kolatorów typu `RuleBasedCollator` odbywa się na podstawie porównywania kolejnych znaków z wykorzystaniem reguł zapisanych w tym obiekcie.
- Reguły te określają cztery porządki:
  `PRIMARY` – rozróżniane są znaki zdefiniowane w zestawie reguł,
  `SECONDARY` – wyodrębnia różnice w znakach akcentowanych,
  `TERTIARY` – rozróżnienie na podstawie wielkości liter,
  `IDENTICAL` – jeśli znaki uznane za takie same według poprzednich trzech reguł są rozróżniane na podstawie kodu.

Porównywanie napisów

- Siła kolatora to stosowany do porównywania porządek. Domyślna siła kolatora to `TERTIARY`.
- Siłę kolatora ustawiamy metodą `setStrength(…)`.
- Zbiór reguł kolatora można odczytać metodą `getRules()` i ustawić metodą `setRules(String)`.
- Reguła to ciąg znaków uporządkowanych relacjami mniejszości (<) dla `PRIMARY`, średnika (;) dla `SECONDARY`, przecinka (,) dla `TERTIARY` i równości (=) dla `IDENTICAL`. Przykład:
  `"…9<a,A<b,B<c,C…"`
  Kolejne reguły łączymy ze sobą za pomocą ampersanda (&). Przykład:
  `"A<B & B<C"`

Porównywanie napisów

- Kolatory można wykorzystać do sortowania tablic napisów metodą `Arrays.sort(…)`. Przykład:

```
String[] tab = …;
Collator col =
  Collator.getInstance();
Arrays.sort(tab,col);
```

Internacjonalizacja aplikacji

- Aplikacja powinna być przygotowana na działanie w różnych środowiskach językowych.
- Nie tylko liczby, daty i czas powinny być prezentowane zgodnie z wymaganiami danej lokalizacji – również komunikacja aplikacji z użytkownikiem powinna spełniać kryteria lokalizacyjne.
- Mechanizmem umożliwiającym odseparowanie języka komunikatów od kodu są w Javie zasoby dodatkowe (ang. *resource bundle*):
  - oparte na tekstowych plikach właściwości `PropertiesResourceBundle` (odseparowanie napisów),
  - oraz oparte na listach obiektów `ListResourceBundle` (odseparowanie dowolnych obiektów).

Internacjonalizacja aplikacji

- Plik właściwości może mieć dowolną nazwę i rozszerzenie `.properties`.
- W pliku takim zapisujemy pary:
  klucz = napis
- Rolę komentarza pełnią linie rozpoczynające się od znaku `#`.
- Gdy do nazwy pliku dołożymy sufiks z nazwą lokalizacji, to utworzymy zbiór napisów dla jakiejś konkretnej lokalizacji.

Internacjonalizacja aplikacji

- Plik o odpowiedniej nazwie możliwie bliskiej bieżącej lokalizacji zostanie odnaleziony i Java pobierze z niego napis przypisany do odpowiedniego klucza.
- Przykład wyboru pliku właściwości, gdy bieżąca lokalizacja to `pl_PL`:
  `Messages.properties`
  `Messages_pl.properties`
  Java wybierze plik `Messages_pl.properties` nie znajdując `Messages_pl_PL.properties`.

Internacjonalizacja aplikacji

- Plik właściwości ze znakami diakrytycznymi należy przetworzyć do pliku w formacie ASCII (zamieniając każdy znak spoza alfabetu angielskiego na sekwencję znaków kodujących szesnastkowo daną literę `\uHHHH` w Unicodzie). Zamiany tej można dokonać automatycznie za pomocą programu `native2ascii`. Przykład:

```
$ native2ascii plik.properties.nat
> plik.properties
```

Internacjonalizacja aplikacji

- W programie odwołujemy się do zewnętrznego pliku właściwości podając nazwę i lokalizację. Przykład:
```
Locale loc = Locale.getDefault();
ResourceBundle msg =
   ResourceBundle.getBundle("name",loc);
```
- Pobranie napisu skojarzonego z podanym kluczem w zasobie `ResourceBundle` dokonujemy metodą `getString`. Przykład:
```
String napis = msg.getString(klucz);
```
- Metoda `getBundle` najpierw szuka obiektów dziedziczących po `ListResourceBundle` a dopiero później plików właściwości.
- Użycie `ListResourceBundle` wymaga pisania kodu ale pozwala kojarzyć z kluczami obiekty innych typów niż `String`.

Literatura

- K.Barteczko: *Java – od podstaw do technologii. Tom 1, część C, rozdział 5: Formatowanie, lokalizacja i internacjonalizacja.* Wydawnictwo MIKOM, Warszawa 2004.
- C.S.Horstmann, G.Cornell: *Core Java – techniki zaawansowane. Wydanie 8. Rozdział 5: Internacjonalizacja.* Wydawnictwo HELION, Gliwice 2009.
- Internationalization (Java Tutorial): http://docs.oracle.com/javase/tutorial/i18n/