Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Localization

Sipos Róbert

siposr@hit.bme.hu

2014. 03. 27.

Bevezetés

- **Localization (L10n)**
  - Egy speciális (politikai, kulturális, földrajzi stb. elven meghatározott) régióhoz való illesztés folyamata
  - A legidőigényesebb része lokalizált tartalmak elkészítése (szövegfordítás, ábrarajzolás)
  - Képek, hangok is eltérőek lehetnek régiónként
  - Dátum-, számformátum, pénznem
- **Internationalization (i18n)**
  - A szöveges elemek nem a programba kódoltak, hanem a forráskódtól különállóan helyezkednek el
  - Új nyelvek támogatása (szám-, dátum-, időformátumok) nem igényel teljes újrafordítást

Localization nélkül

- Beégetett üzenetek

```
public class NotI18N {
    static public void main(String[] args) {
        System.out.println("Hello.");
        System.out.println("Hogy vagy?");
        System.out.println("Viszlát.");
    }
}
```

- A fordító nem programozó…
  - Ha ezeket az üzeneteket más nyelven is meg akarjuk jeleníteni, vagy mi kézzel kicseréljük, vagy leválasztjuk a fő programkódról

Ötlet

- Elkülönített üzenetek egyszerű szöveges állományokban

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

```
greetings = Hello.
inquiry = Hogy vagy?
farewell = Viszlát.
```

- A programban pedig ezeket használhatjuk a `Locale` és `ResourceBundle` (`java.util`) osztályok segítségével

```
[...]
    Locale currentLocale = new Locale(language, country);
    ResourceBundle messages = ResourceBundle.getBundle(
      "MessagesBundle", currentLocale);
    System.out.println(messages.getString("greetings"));
      System.out.println(messages.getString("inquiry"));
      System.out.println(messages.getString("farewell"));
[...]
```

Locale készítése

- A `java.util.Locale` osztály
- Specifikál egy adott régiót

```
import java.util.*;
...
myLocale  = new Locale("de", "CH", "WIN");
myLocale2 = new Locale("hu", "HU");
myLocale3 = new Locale("fr");
myLocale4 = Locale.US;    // léteznek preset locale-ek is
```

- **Nyelvek** kétkarakteres kódjai *kisbetűsen* az ISO-639-es ajánlás szerint
- **Országok** kódjai az ISO-3166 szerinti kétkarakteres *nagybetűs* kódok
- A harmadik argumentumban található ún. variáns az **oprendszert/platformot** jelölheti

Mire használható a Locale

- A `Locale` objektum csak egy azonosító
  - További, locale-érzékeny objektumok használják, melyek a tényleges munkát végzik majd
  - Ezek az objektumok pl. lekérdezhetik a rendelkezésre álló locale-ek listáját, hogy tudják, mivel dolgozhatnak
- Akár minden egyes objektumhoz rendelhetünk külön `Locale`-t
- A JVM-nek mindig van default `Locale`-je, futásidőben lekérdezhető
  - **`Locale.getDefault();`**
- Elosztott rendszernél
  - Vagy a kliens-szerver közi üzenetek locale-függetlenek
  - Vagy a szerver egyes szálai lesznek a kliens locale-évl azonosak

Adatok elkülönítése

- A locale-specifikus objektumokat kulcs-érték párok formájában **`ResourceBundle`**-ökben különítjük el
- Ha egy ilyen objektumra van szükségünk, azt egyszerűen kivesszük a **`ResourceBundle`**-ből; a felhasználó **`Locale`**-jének megfelelő objektumot kapunk vissza
- Előkészületek
  - **Azonosítsuk** a programban a locale-specifikus objektumokat
  - **Csoportosítsuk**, és külön **`ResourceBundle`**-ökben helyezzük el

ResourceBundle – Locale kapcsolat (1)

- Rokon jellegű dolgok összefogása egy **közös alapnévvel**
- Az alapnevet az alosztályok nevei szerint bővítjük

```
ButtonLabel_en_GB
ButtonLabel_en_US_WIN
```

- A megfelelő **`ResourceBundle`** kiválasztása a **`getBundle()`** metódussal történik

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

ResourceBundle – Locale kapcsolat (2)

- A `getBundle()` mindig a **lehető legjobb egyezést** keresi
- Pl. az előző példa (`fr_CA_UNIX` Locale) esetén a következő sort járja végig (a default Locale az `en_US`)

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

- Ha nem talált semmit, **`MissingResourceException`**-t dob
  - Ez elkerülendő: **mindenképp legyen egy alap köteg!** (ez amúgy is jól jön a nyelvi fordításnál)

Alosztályok

- A `ResourceBundle` tulajdonképpen egy absztrakt osztály, két alosztálya létezik
  - **`PropertyResourceBundle`**
    - Külön `properties` fájlokkal támogatott, melyekben kizárólag szöveges, **`String`** formátumú üzenetek tárolhatók
    - Fordítók számára előnyös (nem kell hozzá programozói tudás)
  - **`ListResourceBundle`**
    - Ebben már nemcsak `String`, hanem **bármilyen objektum** tárolható
  - Az absztrakt osztály működése szempontjából mindegy, hogy melyik alosztályt használjuk

PropertyResourceBundle alkalmazása

- Egyszerű szöveges fájlban kulcs-érték párok
- Hozzunk létre egy alapértelmezett fájlt
  - Egyszerű text fájl
  - **<ResBundlenév>.properties** névre hallgat
- Hozzuk létre a további locale-ekhez használandó szöveges információk fájljait
  - **<ResBundlenév>_<nyelv>_<régió>_<variáns>.properties** névre hallgató fájlok lesznek

```
#LabelsBundle.properties (default)
s1 = computer
s2 = disk
s3 = monitor
s4 = keyboard
```

```
# LabelsBundle_hu.properties
s1 = Számítógép
s2 = Lemez
s3 = Monitor
s4 = Billentyűzet
```

ListResourceBundle alkalmazása

- Egy-egy osztályt kell létrehozni minden támogatott Locale-hez
- Az osztályban egy kettős `Object` tömbben kulcs-érték párok szerepelnek

```
class ButtonLabel_en extends ListResourceBundle {

    public Object[][] getContents() {
        return contents;
    }

    static final Object[][] contents = {
        {"OkKey", "OK"},
        {"CancelKey", "Cancel"},
    };
}
```

ResourceBundle létrehozása

- **`ResourceBundle`**-re a **`getBundle()`** metódussal szerzünk referenciát

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

- A **`getBundle()`** először egy, az alapnévnek és a **`Locale`**-nek megfelelő nevű *osztályt* keres
  - Találat esetén `ListResourceBundle` jön létre
- Ha nem talált ilyen osztályt, akkor `properties` fájlt keres
  - Ha ezt megtalálta, létrehoz a fájl alapján egy `PropertyResourceBundle` objektumot
  - Ha ezt sem találja, akkor `MissingResourceException`-t dob

ResourceBundle – Lokalizált szöveg kinyerése

- `PropertyResourceBundle` esetében csak `String`-ekkel dolgozhatunk
- A **`getString()`** metódusnak a kiválasztott elemet azonosító kulcsot adjuk át, megkapjuk a hozzátartozó értéket

```
String value = labels.getString("s1");
```

- A visszakapott `String` a `Locale`-nek megfelelő, vagy a default lesz
- Ha a megadott kulcshoz nincs érték, `MissingResourceException` keletkezik
- A `getString()` alkalmazható `ListResourceBundle`-re is, de ha a kulcs mögötti érték nem `String`, `ClassCastException`-t kapunk

ResourceBundle – Objektumok kinyerése

- A **`getObject()`** metódussal szerezhetjük meg a szükséges lokalizált objektumokat, ahol a kívánt objektum kulcsát adjuk meg

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

- Mivel a `getObject()` `Object`-et ad vissza, azt a tényleges használathoz a megfelelő típussá kell kasztolni

Összes kulcs

- Esetenként szükség lehet a köteg összes kulcsára
- A **`keySet()`** egy **`Set<String>`**-et ad vissza a kulcsokból, amin aztán végiglépkedhetünk

```
ResourceBundle labels =
ResourceBundle.getBundle("LabelsBundle", currentLocale);
for (String key : labels.keySet()) {
    String value = labels.getString(key);
    System.out.println("key = " + key + ", " + "value = " + value); }
```

További lehetőségek

Számok formázása

- **`java.text.NumberFormat`**: egyszerű típusok formázása

```
Double amount = new Double( 345987.246 );
NumberFormat numberFormatter =
NumberFormat.getNumberInstance( currentLocale );
String amountOut = numberFormatter.format( amount );
System.out.println( amountOut + " " + currentLocale.toString() );
```

Saját számformátum

- **`java.text.DecimalFormat`**: saját formázási minta adható meg
  - Pl.: `#,#00.0#` → `1,234.56`

```
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(pattern + " " + output + " " +
            loc.toString());
}
```

- **`java.text.DecimalFormatSymbols`**: saját szimbólumokat használhatunk (tizedespont helyett vessző, aposztróf helyett space csoportosító)

Dátumok, idő

- **`java.text.DateFormat`**: előre definiált dátum- és időformázások

```
DateFormat dateFormatter =
   DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
Date today = new Date();
String dateOut = dateFormatter.format(today);
System.out.println(dateOut + " " + currentLocale.toString());
```

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

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

Dátum, idő – saját minták megadása

- **`java.text.SimpleDateFormat`**: előre definiált dátum- és időformázások

  pl. „`yyyy.MM.dd. HH:mm:ss`” → 2001.07.04. 12:08:56

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

- **`java.text.DateFormatSymbols`**: saját szimbólumok megadásának lehetősége

Printf

- A C-szerű nyelveknél megszokott “%.2f”-hez hasonló formázásra a **Java 5.0 óta** van lehetőség
- A `java.util.Formatter` osztály [5.0] `format` metódusait használja

```
Double d = new Double(132.567898);
Locale hu = new Locale("hu", "HU");
Locale en = new Locale("en", "US");
System.out.printf( hu, "%1$.2f", d); // 132,57
System.out.println();
System.out.printf( en, "%1$.2f", d); // 132.57
System.out.println();
Calendar c = Calendar.getInstance();
System.out.printf(hu,"%1$tY. %1$tB. %1$td.",c); // 2005. október 27.
```

- A String osztály is rendelkezik egy statikus formázó metódussal

```
System.out.println(String.format(hu, ”N=%d”, 42));
```

Összetett üzenetek formázása (1)

- Template-eket veszünk fel egy `ResourceBundle`-ben

```
ResourceBundle messages =
    ResourceBundle.getBundle("MessageBundle", currentLocale);
```

- A szokásos kulcs-érték párokat használjuk

```
template = {2,date,long} napon {2,time,short} órakor, \
        {1,number,integer} baleset történt {0} városban.
city = Budapest
```

- Ugyanez megtehető `ListResourceBundle`-lel is

```
public class MessageBundle extends ListResourceBundle{
...
private final Object[][] contents = {
    { "template", "{2,date,long} napon {2,time,short} órakor,
      {1,number,integer} baleset történt {0} városban." },
    { "city", "Budapest"} };
}
```

Összetett üzenetek formázása (2)

- Üzenet-argumentumok felvétele

```
Object[] messageArguments = {
    messages.getString("city"),   // 0. elem
    new Integer(7),  // 1. elem
    new Date()   // 2. elem
};
```

- Formatter létrehozása

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

- A formázott üzenet előállítása

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

Karakterek osztályzása

- Unicode: a világ nyelveinek összes karaktere egyedi azonosítóval ellátva 16 biten kódolva
  - Platform-, program- és nyelvfüggetlenül
- A Java Unicode-ot használ – az egyedi azonosítók kihasználása
- A Character osztály metódusaival osztályozhatjuk a karaktereket
  - `isDigit`
  - `isLetter`
  - `isLetterOrDigit`
  - `isLowerCase`
  - `isUpperCase`
  - `isSpaceChar`
  - `isDefined`

String-komparálás

- A `String.compareTo()` sajnos nem törődik a nemzeti karakterek igazi sorrendjével (a karakterkódok nem ezt követik)
- A `java.text.Collator` segítségével ez a probléma megoldódik

```
Collator myCollator = Collator.getInstance(new Locale("en", "US"));
System.out.println(myCollator.compare("abc", "def"));
System.out.println(myCollator.compare("rtf", "rtf"));
System.out.println(myCollator.compare("xyz", "abc"));
```

- A `RulebasedCollator`-ben rendezési szabályokat is megadhatunk

```
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");

RuleBasedCollator enCollator =
    new RuleBasedCollator(englishRules);
```

Unicode lehetőségek

- Kevés text editor használja egyelőre a Unicode-ot, mint kimeneti formátumot
  - Ilyenkor escape szekvenciákkal állíthatunk elő Unicode karaktereket

```
String str = "\u00F6";
char c = '\u00F6';
Character letter = new Character('\u00F6');
```

- További lehetőségek Unicode és a `java.text` csomag használatával
  - Héber írás (visszafelé a `Bidi` (bidirectional) osztály használatával)
  - Kínai karakterek írása
  - stb.