CSE 403
Lecture 22

Localization

Reading: *Core Java, Volume II: Advanced Features, 8ed*
by Cay Horstmann, Ch. 5

*Agile Web Development with Rails, 3ed*
by Ruby/Thomas/Hansson, Ch. 13

slides created by Marty Stepp
http://www.cs.washington.edu/403/
i18n and L10n

- **internationalization** ("i18n"): Process of designing software so that it can be adapted to various languages and regions.
  - done once per product (ideally); updated as code is added

- **localization** ("L10n"): Process of adapting/translating internationalized software for a specific region or language.
  - done once per locale; each locale is updated as text is added

- **globalization** ("g11n"): i18n + L10n
  - Less commonly used term, but many companies use it.
Who cares?

- Why should a team want to internationalize / localize its app?
  - reach a wider audience
  - make more $$$

- Is it worth it to localize?
  - May need to evaluate cost/benefit:
    - What fraction of our users speak that language?
    - Are they also fluent in English?
    - Are they already able to use the site now?

- open-source software is often translated for free by community
  - Maybe you can post your code and let them do it ...
• **Unicode**: Standard for storing, encoding, numbering over 107,000 chars from > 90 languages.
  
  – created in 1991 by non-profit Unicode Consortium
  – standard character ↔ integer mappings
  – Translation Formats (UTF-*) to store chars as bytes
  – supported by languages (Java,.NET,Python), browsers
  
  – important for localization because it defines int'l chars and encodings we will use to present localized text
Character encodings

• **ISO-8859-1**: ANSI, 8-bit (extended ASCII)
  – backward-compatible; simple; mostly English-only

• **UTF-8**: 1 byte for all ANSI chars, which have the same code values as in standard ASCII; up to 4 bytes for other chars

• **UTF-16**: uses 2 bytes for almost all characters, and 4 bytes to encode certain special characters

• Code files, web pages may specify or be saved with encoding:

```html
<html>
  <head>
    <title>CSE 403, Winter 2048</title>
    <meta charset="utf-8" />
  </head>
</html>
```
• **locale**: A geographic/cultural location targeted for localization.
  
  A locale consists of:
  – a language (e.g. English)
    • often expressed as an [ISO-639-1](https://en.wikipedia.org/wiki/ISO_639-1) code: de, en, fr, ja
  – a location or variant (e.g. United States, UK)

• Why isn't it enough to specify just the language?
  – different locations may use different conventions, spelling, etc.
    • "color" (US) vs. "colour" (UK)
    • "localize" (US) vs. "localise" (UK)
  – some locations use dialects of a given language
  – other differences (dates, currency, numbers, time zone, etc.)
Differences between locales

- **language**
  - English vs. German

- **spelling**
  - "color" vs. "colour"

- **slang**
  - "line" vs. "queue"

- **number formatting**
  - telephone number format: (206) 949-0504 vs. +1.206.949.0504
  - decimal separator, digit groups

- **currency units/formatting**
  - $123.45 vs. 123,45€

- **date formatting**
  - 3/14/10 vs. 2010/Mär/14

- **keyboard shortcuts**

- **text direction**
  - left-to-right ("LTR") vs. right-to-left ("RTL"), AKA "bidi" issues

- **multimedia**: spoken audio; video subtitles
Right-to-Left (RTL)

- some languages begin lines on the right side and go left
  - Arabic, Farsi/Persian, Hebrew, Kurdish, Punjabi, Somali, ...
  - hello vs. שלום

- often handled by supplying separate .css files for RTL locales

- can lead to lots of subtle UI bugs based on coders' LTR assumptions
Localization gotchas

- Some languages (e.g. German) use long words
  - buttons/labels get too wide for space provided
- Some Unicode characters look like ASCII ones
  - U+00A0 "non-breaking space" character
  - "-" vs. U+2014 "em-dash" —, U+2013 "en-dash" —
- some fonts don't have all characters
  - but a smart OS can use font substitution
- regular expressions / text searches may not match i18n input
  - ex. \w "word boundary" doesn't match Unicode word delimiters
- web server might return text that has not been localized (Ajax)
Things to avoid for i18n

• Don't hard-code widths/heights in CSS or GUI

• Avoid images that look like text.

• Avoid using symbols that have no meaning in other locales.
  – USA "STOP sign"
  – Hand up for "Wait"

• (if localizing to RTL locales) Avoid hard-coding the notion that "left" means "start" and "right" means "end".
  – example: Left for "Back", right for "Forward"
  – example: Left for "less", right for "more"
How are i18n/L10n done?

- **developers** *internationalize* the app's code
  - pull all strings out of code and into separate resource files
  - call methods that localize/format strings, numbers before printing
  - use libraries (e.g. *gettext*) to help localize messages

- **localizers** (maybe not programmers) *localize* the app's text
  - often hired to localize an app for a particular locale at a time
  - *desktop apps*: possibly compile a different binary for each locale
  - *web app*: look up localized strings when generating each page

- **model-view separation** is very important for i18n/L10n
Perils of poor localization

• word "Okay" could be translated as "so-so" or "mediocre"

• << and >> , when used as "arrows", can confuse some users whose languages use << and >> as quotation marks

• product's name or ad could translate poorly
  – Microsoft's "Bing" can translate to "disease" in Chinese
  – McDonald's pictures-only billboard in Saudi Arabia

• product could offend users from other countries
  – an online dating site that allows users under 16 to register?
  – an online auction site that has bidding end on a holy day?
Poorly localized messages

- "Drop your pants here for best results."
- "We take your bags and send them in all directions."
- "Ladies may have a fit upstairs."
- "Teeth extracted by latest methodists."
- "Please leave your values at the front desk."
- "No smoothen the lion."
- "If you consider our help impolite, you should see the manager."
- "Our wines leave you nothing to hope for."
- "It is forbidden to enter a woman, even if dressed as a man."
- "Fur coats made for ladies from their own skin."
- "Specialist in women and other diseases."
- "Ladies, leave clothes here and spend afternoon having good time."
- "We regret that you will be unbearable."
- "When passenger of foot heave in sight, tootle the horn. Trumpet him melodiously at first, but if he still obstacles you then tootle him with vigor."

- dry cleaning, Tokyo
- Scandinavian airport
- dry cleaning, Bangkok
- dentist, Hong Kong
- hotel, Paris
- zoo, Czech
- hotel, Athens
- Swiss restaurant
- Bangkok temple
- Swedish furrier
- doctor, Rome
- laundry, Rome
- hotel, Bucharest
- car rental, Tokyo
Automated translation

- Why hire localizers when automated translators exist, such as Google Translate and BabelFish?

  - From course syllabus:
    "There is no textbook, but there will be reading assignments throughout the quarter that will be posted to the course web site to print or read online. For many of the reading assignments, we will assign questions posted online for you to answer about the reading. You will submit your answers to these questions online. These will be part of your course grade and will not be accepted late."

  - Translated to Chinese by BabelFish, then back to English:
    "Without the textbook, but will have will be posted to the route website, in on-line printing or reads in quarter reading assignment. For many reading assignments, we the assignment problem on-line will post, for you can reply that about studies. You on-line will submit your answer to give these questions. These will be a your route rank part, and after them, will not be accepted."
Android localization

- Android apps store resources in `res/` folder
  - text strings go in `strings.xml` files:
    - `res/values/strings.xml` (defaults)
    - `res/values-en/strings.xml` (English)
    - `res/values-fr/strings.xml` (French)

- testing locales:
  - **(on a real Android device)**
    Home → Menu → Settings → Locale & text → Select locale
  - **(on the emulator using adb)**
    `adb -e shell`
    # setprop persist.sys.language [language code]; setprop persist.sys.country [country code]; stop; sleep 5; start
strings.xml example

<!-- res/values/strings.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="greeting">Hello!</string>
  <string name="login">User %1 logged in.</string>
</resources>

<!-- layout XML file that uses string in View -->
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/greeting" />

// in-app Java code that grabs a resource string
Resources res = getResources();
String greet = res.getString(R.string.greeting);
String msg = String.format(
    res.getString(R.string.login), userName);
Other resources

• Images are in res/drawable
  – res/drawable/company-logo.png (English)
  – res/drawable-fr/company-logo.png (French)

<!-- layout XML file that uses image in View -->
<ImageView
  android:layout_height="wrap_content"
  android:layout_width="wrap_content"
  android:src="@drawable/company-logo" />

• Layouts are in res/layout
• Menus are in res/menu
• Colors are in res/values/colors.xml
IDE resource support

- Many IDEs (Eclipse) help you graphically create/edit resources.
Locales in standard Java

• `java.util.Locale` class represents a locale on the system

```java
Locale here = Locale.getDefault();
Locale swiss = new Locale("de", "CH");
System.out.println(swiss.getDisplayName());
System.out.println(swiss.getDisplayName(Locale.GERMAN));
Locale.setDefault(Locale.ITALIAN);
```

• many other classes have methods to get supported `Locale`

```java
Locale[] locs = DateFormat.getAvailableLocales();
```

• many other classes have methods that accept a `Locale`

```java
NumberFormat euro =
    NumberFormat.getCurrencyInstance(Locale.US);
```
Resource files

**resource**: An external file (not code) containing a list of text messages localized for a particular locale.

benefits:
- putting messages together makes sure you don't forget any
- easier to add another language later
- no locale-based logic in program code
- non-programmers can work on localization

- given file names that contain locale info
  - e.g. PracticeIt_de_DE.properties
Resource bundles in Java

- standard (non-Android) Java's resource file format:
  - a "properties file", full of `name = value` pairs
    named something like `MyProgram_de_CH.properties`

  ```java
  computeButton=Rechnen
  colorName=black
  defaultPaperSize=210x297
  ```

- `java.util.ResourceBundle` class reads resource files

  ```java
  ResourceBundle bundle = ResourceBundle.getBundle(
     .bundleName, locale);
  String buttonText = bundle.getString("computeButton");
  ```

- can bundle objects by extending `ResourceBundle`, but don't
IDE resource support

- Many IDEs (Eclipse) help you graphically create/edit resources
Formatting numbers

- `java.text.NumberFormat` formats numbers for a locale

```java
Locale loc = Locale.getDefault();
NumberFormat currFmt =
    NumberFormat.getCurrencyInstance(loc);
String result = currFmt.format(1234.56);  // 1.234,56€

// don't use Double.parseDouble, etc.
String line = scanner.nextLine();  // user input
Number input = fmt.parse(line.trim());

- throws ParseException if text is in invalid format (need to trim)
```
### Formatting dates

- Differences in how to display dates across locales:
  - names of the months/days (Monday vs. Lundi)
  - ordering of days (what day does a week start/end)
  - relative order of y/m/d (3/14/2010 vs. 2010/Mar/14)
  - time zone (usually offset from UTC/GMT)
  - 12 vs. 24 hour time (5:00 PM vs. 17:00)

- **java.text.DateFormat** formats dates
  - styles: `DateFormat.DEFAULT`, `FULL`, `LONG`, `MEDIUM`, `SHORT`

```java
DateFormat fmt = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.SHORT, locale);
String s = fmt.format(new Date());
Date d = fmt.parse(dateText.trim()); // parse a date
```
Formatting currencies

• currencies are represented by [ISO-4217](#) currency identifiers
  – examples: USD, GBP, EUR, JPY, CNY, INR, RUB
  – programming languages don't know exchange rates between currencies (can't tell you how many Euros equals $100.00)
  – but facilities exist for displaying a variable as a currency amount

• `java.text.NumberFormat` has currency objects (instances)

```java
NumberFormat dollar =
    NumberFormat.getCurrencyInstance(Locale.US);
NumberFormat euro =
    NumberFormat.getCurrencyInstance(Locale.GERMANY);
euro.setCurrency(Currency.getInstance("EUR"));
String s = euro.format(123456.78);
```
String collation

- **collation**: Locale-dependent relative ordering of strings.
  - *String's compareTo* method just goes by ASCII/Unicode value
  - doesn't work well for special characters: é, Å, æ, ß, ™
    - one locale might want Å to come between A and B
    - another might want Å to come after Z...
  - normalization/decomposition: turning ™ into "™", é into "e´"

- **java.text.Collator** class collates strings for a given locale
  - implements Comparator interface
  - **collation strengths**: PRIMARY, SECONDARY, TERTIARY

```java
Collator coll = Collator.getInstance(locale);
coll.setStrength(Collator.TERTIARY);
if (coll.compare(a, b) < 0) { ... // a comes before b }
```
String/text formatting

• When dealing with strings that insert variables' values, it is best practice to make them use `printf / String.format / etc.`
  – This gives one complete sentence/string for the localizers.
  – Easier to understand context and translate properly.

• `java.text.MessageFormat` formats localized strings

```java
String format = "On {2}, a {0} caused {1} damage."
String msg = MessageFormat.format(message,
    "hurricane", 10.0E8,
    new GregorianCalendar(1999, 0, 1).getTime());
// On 1/1/99 12:00 AM, a hurricane caused 100,000,000 damage.

format = "On {2,date,medium}, a {0} caused {1,number,currency} damage.";
// On Jan 1 1999, a hurricane caused $100,000,000 damage.
// can specify placeholder format: number, time, date, ...
```
Localization in Ruby/Rails

- **I18n module** has `translate(t)` and `localize(l)` methods

  ```ruby
  t 'store.title'  # translate the given string id
  l Time.now       # localize the current time
  ```

- you can also get/set the following `I18n` properties as necessary:
  - `load_path`, `locale`, `default_locale`, `exception_handler`, ...

- setting locale based on URL subdomain:
  ```ruby
  # de.myapp.com -> set I18n.locale to "de"
  I18n.locale = request.subdomains.first
  ```

- setting locale based on HTTP request
  ```ruby
  # request is "de" -> set I18n.locale to "de"
  I18n.locale = request.env['HTTP_ACCEPT_LANGUAGE']
               .scan(/^[a-z]{2}/).first
  ```
Localization resource files

- each language gets a `.yml` file of hierarchical key/value pairs:

  ```
  en:
    hello: "Hello, world!"
  store:
    title: "Jim's Meat Market"

  es:
    hello: "Hola, mundo!"
  store:
    title: "Mercado carne de Jim"
  ```

- views can refer to a key from the resource file:

  ```erb
  # app/views/home/index.html.erb
  <h1><%= t :hello %></h1>
  <p><%= flash[:store.title] %></p>
  <h2><%= l Time.now %></h2>
  ```