Internationalization & Localization
Definitions

- **Internationalization:** preparing an application for translation.
  - Sometimes written `i18n`
  - Separate language-specific portion from rest of the code

- **Localization:** Translate text portion to a particular language
  - Sometimes written `l10n`
  - Usually done by non-programmers
  - Non-trivial
  - Requires knowledge of language & cultural/national standards
I18N challenges

- Different languages
  - Encodings (e.g., Unicode)
  - Fonts
- Text direction (left-to-right vs. right-to-left)
- Conventions
- Parameterization
  - `player.getName() + “ made a book of ” + rankName + “s.”`
- Plurals
- Text length
  - Affects layout
Unicode concepts

- Each letter is represented by a “code point”
  - H => U+0048
- Fonts are mappings from code points to “glyphs”
  - Glyphs are pictures of characters
- There are many possible encodings of Unicode code points
  - UTF-16 (UCS-2) is the Java standard (16bits = 2bytes/char)
  - UTF-8 is 1 byte/char (efficiently stores ASCII subset)
  - UTF-7 and UTF-32 (UCS-4) are less common
  - All can store any Unicode code point
Conventions

- **Number formats**
  - US/UK: 72,530.55
  - France: 72 530,55
  - Germany: 72.530,55

- **Date formats**
  - US: Month/Day/Year
  - Everywhere else: Day/Month/Year

- **Colors & Icons**
  - Different cultural associations
    - E.g., white = purity (west), death (china)
Locales

- **Two components:**
  - Language
  - Country (optional)
  - Charset (optional, rarely used)

- **Examples:**
  - en_US vs. en_GB
  - zh_CN (or zh_HANS) vs. zh_TW (or zh_HANT)

- **Setting locale (for testing)**
  - VM options: `-Duser.country=CA -Duser.language=fr`
Structuring code for I18N

- Separate the text from the code
  - Java uses “resource bundles”
  - Separate resource for each locale
  - Support for new languages does not require recompilation

- Use libraries to handle formatting of:
  - Numbers, dates, currency, ...
Resource Bundle

- **Creation:**
  - In source path with same package
  - Add all of the locales that you want to support

- **Usage:**
  - Static variable in class that wants to use it
  - `private static ResourceBundle resources = ResourceBundle.getBundle("ninja.zilles.Resources");`
    - Optional second parameter to over-ride locale
  - `resources.getString("nameOfString")`
What if your text includes variables?
  - Different languages might put them in different places
  - “Mary’s hand” vs. “La main de Mary”

Provide templates that can be parameterized
  - o.s.hand={o}’s hand // en
  - o.s.hand=La main de {o} // fr
  - MessageFormat.format(resources.getString("o.s.hand"), name);
Plurals (ChoiceFormat)

- Different sentence structure based on number
  - “there are no cats”, “there is one cat”, “there are 12 cats”
- private static double[] catLimits = {0,1,2};
  private static String [] catStrings = {
    resources.getString("no.cats"),
    resources.getString("one.cat"),
    resources.getString("multiple.cats")
  };
- private static ChoiceFormat catChoice = new ChoiceFormat(catLimits, catStrings);
- String template = catChoice.format(numCats);
  String catString = MessageFormat.format(template, numCats);
Number Formatting (existing libraries)

static public String displayNumber(Locale currentLocale) {
    NumberFormat numberFormatter =
        NumberFormat.getNumberInstance(currentLocale);
    Double amount = new Double(345987.246);
    String formattedAmount = numberFormatter.format(amount);
    return formattedAmount + "   " + currentLocale.toString();
}

345 987,246  fr_FR
345.987,246  de_DE
345,987.246  en_US

NumberFormat.getNumberInstance() gets default locale
Currency Formatting (existing libraries)

```java
static public String displayCurrency(Locale currentLocale) {
    Double currencyAmount = new Double(9876543.21);
    Currency currentCurrency =
        Currency.getInstance(currentLocale);
    NumberFormat currencyFormatter =
        NumberFormat.getCurrencyInstance(currentLocale);

    return currentLocale.getDisplayName() + " , " +
        currentCurrency.getDisplayName() + ": ": " +
        currencyFormatter.format(currencyAmount));
}
```

French (France), Euro: 9 876 543,21 €
German (Germany), Euro: 9.876.543,21 €
English (United States), US Dollar: $9,876,543.21
Date Formatting (existing libraries)

```java
DateFormat dateFormatter =
    DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
Date today = new Date();
return dateFormatter.format(today) + " " + currentLocale.toString();
```

<table>
<thead>
<tr>
<th>Style</th>
<th>U.S. Locale</th>
<th>French Locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>Jun 30, 2009</td>
<td>30 juin 2009</td>
</tr>
<tr>
<td>SHORT</td>
<td>6/30/09</td>
<td>30/06/09</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>Jun 30, 2009</td>
<td>30 juin 2009</td>
</tr>
<tr>
<td>LONG</td>
<td>June 30, 2009</td>
<td>30 juin 2009</td>
</tr>
<tr>
<td>FULL</td>
<td>Tuesday, June 30, 2009</td>
<td>mardi 30 juin 2009</td>
</tr>
</tbody>
</table>
To Do

- **For Code Review:**
  - Part 1: Internationalize your GoFish implementation
    - Support English/US and **one other language**.
    - Correctly handle plurals
  - Part 2 coming on Thursday