Internationalization & Localization
ISA relationship – which is NOT true?

- A. In Java a class can extend multiple classes
- B. In UML Case Diagram uses a line with an triangular arrow
- C. In Java a class can extend multiple interfaces
Android App Development?

- A) Yes
- B) A bit (<10 hours)
- C) A tiny bit (< 5 hours)
- D) None
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)
    - 4
Locales

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

- **Examples:**
  - `en_US` vs. `en_GB`  
    `car.storage="trunk","boot"`
  - `zh_CN` (or `zh_HANS`) vs. `zh_TW` (or `zh_HANT`)
Testing Locales

- Setting locale (for testing)
  - VM options:  -Duser.country=CA -Duser.language=fr
  - `Locale.setDefault( localeobject )`

- `Locale(String language)`
  `Locale(String language, String country)`
  `Locale(String language, String country, String variant)`

- See Docs for useful constants `Locale.US`
- [https://docs.oracle.com/javase/7/docs/api/java/util/Locale.html?is-external=true](https://docs.oracle.com/javase/7/docs/api/java/util/Locale.html?is-external=true)
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("mypackage.pack.resoures");`
    - Optional second parameter to over-ride local
  - `resources.getString("nameOfString")`
MessageFormat

- 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
  - 0.s.hand=${0}’s hand  // en
  - 0.s.hand=La main de {0}  // fr
  - MessageFormat.format(resources.getString("0.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)

```java
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
String displayCurrency(Locale locale) {
    Double amount= new Double(9876543.21);

    currency= Currency.getInstance(locale);

    formatter= NumberFormat.getCurrencyInstance(locale);

    return Locale.getDisplayName() + " , " +
        currency.getDisplayName() + " : " +
        formatter.format(amount));
}
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</td>
</tr>
<tr>
<td>juin 2009</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>