Externalizing Resources

- If you think about it, certain application changes should *not* really require a recompile:
  - Strings (labels, tooltips)
  - Icons, sounds
- Further, in line with many interaction design principles, *internationalization* (or *i18n* for short) is extremely desirable, and is in fact required in many contexts
- Thus, an easy mechanism for keeping these *resources* apart from the code, so that they can be changed easily, is extremely useful — and Java has one

What "Resources" Are

- The idea behind a "resource" is that it is a distinct information package, separate from the source code, but which the source code expects to be available
- The source code accesses this resource through some uniform interface that retrieves the information needed by the application
- The interface should also provide for transparent *swapping* of resources — for example, switching from English to Spanish labels — without changing any of the source code (say, as a user preference)

Resources in Java: *java.util.ResourceBundle*

- Java's abstraction for resources is *ResourceBundle*
- *ResourceBundle* works like a map or dictionary: given a key string, it will return a string or object that has been mapped to that key
- *ResourceBundle* "automagically" finds the appropriate file containing the strings that you want, based on its name and its location on the classpath
- If needed, you can even subclass *ResourceBundle* to retrieve its information in a different manner

Internationalization in Java: *java.util.Locale*

- A part of *ResourceBundle*'s algorithm for determining the resource file to use is the *Locale* to use
- A *Locale* is Java's abstraction for the information that varies based on the user's geographical, political, or cultural region: specifically, a *Locale* holds a *country* and a *language*, both represented in accordance with ISO standards (see the *Locale* Javadoc API for details)
- The Java virtual machine sets up the default *Locale* automatically; customization can be done by modifying the *user.language* and *user.country* system properties

Getting "Externalized"

- To achieve externalization (and thus, i18n) in Java, start by making some external preparations:
  - Prepare the resource *properties* file(s) — these are simple key-value text files consisting multiple lines with the format *key=value*
  - Arrange for the file(s) to be in your app's classpath
- In the code, do the following:
  - Provide an easy-to-access mechanism for determining the *ResourceBundle* to use and for retrieving strings from this bundle
  - Stop using hardcoded strings in your code; instead, every time you need a string (say, for a label, menu item, or button), access the *ResourceBundle* instead
  - Some development environments have tools that assist in this process (e.g., Eclipse)

Neat Externalization Tricks

- To support internationalization, create multiple properties files, following a well-defined naming scheme (gory details are in the *ResourceBundle* Javadoc API, under the *getBundle()* method) — *and that's it*
  - *ResourceBundle* will automatically and transparently choose the appropriate resource file based on the current language and country
- Externalized strings don't have to be visible labels: they can be strings for anything, so this mechanism is potentially useful for anything else in your application that may vary based on the country or language