Exercises: Properties Files and I18N

1. Create a form that collects a first name, last name, and email address. Use a properties file for the prompts and button label. There is no need to connect the form to any real action controller or have any navigation rules. Also, you do not need to use properties with parameters (e.g, {0}, {1}): simple strings are sufficient.

- Note: properties files go in src in your Eclipse project. Also, typing in the <application> and <resource-bundle> tags in faces-config.xml is tedious. I recommend you just copy these tags from faces-config in my "properties" project, then change the file name and variable name to match your app. Or, you can use Eclipse auto-completion.

2. Create versions of the properties file in two other languages such as Spanish (_es), French (_fr), Italian (_it), or German (_de). If you don't know any other languages, try using Google Language Tools (http://translate.google.com/) or Babel Fish (http://babelfish.com/) to do the translations. Note that although Java and JSF can easily represent other character sets since Java uses Unicode internally, the Eclipse property editor is not so good at that. So, if you cut and paste Chinese or Russian or Hindi from translate.google.com, Eclipse will store it as explicit Unicode characters (\u092F, etc.) It will display properly in the browser, but if you speak that language, it will be virtually impossible for you to edit the text directly in Eclipse.

3. Internationalize your application by having the Locale automatically selected based on the browser language settings. Remember to wrap the contents of the page between <f:view locale="..."> and </f:view>, and to provide the browser locale to f:view:
<f:view locale="#{facesContext.externalContext.requestLocale}">
If you want to just see what your browser is sending for the Locale, you can do this:
<h1>Browser Locale: #{facesContext.externalContext.requestLocale}</h1>

4. Make a new form that uses the same property files, but this time the locale should be taken from a bean. Return a Locale at random from your bean method, so that every time the user reloads the page, the language could potentially cycle among the languages that you support. The Locale class takes a locale name (e.g., "es" or "es_mx") as a constructor argument. See the online API for java.util.Locale for details, but here is a quick example:

```
public Locale getRandomLocale() {
  if (Math.random() > 0.5) {
    return(new Locale("en")); // Or a constant with that value
  } else {
    return(new Locale("es")); // Or a constant with that value
  }
}
```

Now, you would get the locale from the bean instead of from the browser:

```
<f:view locale="#{userPrefs.randomLocale}">
```