12 - Localization

< Previous | Next >

- A resource bundle file could be a properties file or a class file.
- The abstract class ResourceBundle has two subclasses: PropertyResourceBundle and ListResourceBundle.
- A PropertyResourceBundle is backed by a properties file. A properties file is a plain-text file that contains translatable text. Properties files are not part of the Java source code, and they can contain values for String objects only. If you need to store other types of objects, use a ListResourceBundle instead.
- The ListResourceBundle class manages resources with a convenient list. Each ListResourceBundle is backed by a class file. You can store any locale-specific object in a ListResourceBundle. To add support for an additional Locale, you create another source file and compile it into a class file.
- Remember that a resource bundle is a properties file. A properties file is a plain text file that contains key-value pairs.
- Valid termination characters are: `=` , `:` , or white space character other than a line terminator. These may be escaped with a preceding backslash character.
- Comments in a properties file start with a `#` or `!` .
- Valid ways to create a Locale: `Locale.getDefault()` , `Locale.US` , `new Locale("ru", "RU")` , `Locale myloc = new Locale.Builder().setLanguage("hinglish").setRegion("IN").build();`
- The above code is trying to create a resource bundle for a specific locale i.e. `hinglish_IN` . To create this bundle, at least one of `mymsgs_hinglish_IN.properties` , `mymsgs_hinglish.properties` , and `mymsgs.properties` must be present in the classpath. Since none of these files is not available, the resource bundle cannot be created. An exception will therefore be thrown when you call getBundle().
- Remember that when a resource bundle is not found for a given locale, the default locale is used to load the resource bundle. Fall back option is language without country! As a last resort, it will try to load a resource bundle with no locale information. If this is not available it will throw an exception at runtime.
- Resource Bundles, which are nothing but appropriately named properties files, are used along with the Locale (i.e. country and language) information to format Date, Currencies, and text messages in Locale specific manner.
- mymsgs.properties is the base file for this resource bundle. Therefore, it will be loaded first. Since the language and region specific file is also present (`_en_UK`), it will also be loaded and the values in this file will be superimposed on the values of the base file.Remember that if there were another properties file named `mymsgs_en.properties` also present, then that file would have been loaded before `mymsgs_en_UK.properties` .
- ResourceBundle has a getObject and getString and getStringArray method: `Object obj = rb.getObject("key1");` and `String[] vals =  rb.getStringArray("key2");`
- There is no getValue method in ResourceBundle.
- Keys in resources are always Strings. So you cannot use an int to get value for a key. `Object obj = rb.getObject(1);`

 # I18N:

- i18n indeed stands for Internationalization, it is not done automatically. You have to write code to internationalize your output.
- You should use Locale and formatter objects such as NumberFormat and DateFormat to generate locale specific output.
- When not passed to the getInstance() method, the default Locale is used, which is same as the one set by the operating system. If you want to change it, (for example, if you want to generate French format on a US machine), you must create a new Locale("fr", "FR") object and use the following methods to get an appropriate NumberFormat or DateFormat instance:
  - NumberFormat: `NumberFormat getInstance(Locale locale)`
  - DateFormat: `DateFormat getDateInstance(int style, Locale locale)` Note that DateFormat does not have `getInstance(Locale locale)` method.

< Previous | Next >