Localización

# Temas

* Leer y fijar el local usando el objeto Locale
* Construir un resource bundle para cada local
* Cargar un resource bundle en la aplicación
* Formatear texto para localización usando NumberFormat y DateFormat

# Introducción

Las computadoras y el software se han vuelto indispensables ahora y son usados en todo el mundo para actividades humanas. Para cualquier software que sea relevante y útil para estos usuarios, necesita ser localizado. El proceso en el cual adaptamos al software al lenguaje local y sus costrumbres se llama Localization (localización). Un locale representa el conjunto de lenguajes, cultura, números, tipo de cambio, moneda, etc. de un país.

Java provee buen soporte para hacer software localizable; cubriremos los temas relacionados en detalle en este capítulo. También, Java soporta Unicode, y muchas computadoras tienen las fuentes necesarias para mostrar texto en múltiples idiomas, es nuestro trabajo adaptar el software a los diferentes locales. Por ejemplo, la localización no solo significa imprimir el texto para el local, también significa usar audio o video clips para ese locale. Por lo tanto, aspectos relacionados con fechas, tiempo, moneda local, etc., deben ser considerados también.

En este capítulo aprenderás como localizar tu software. La localización implica crear resource-bundles para diferentes locales, tanto como para hacer software que cuida la cultura adaptándose a diferentes locales. Ahora te mostraremos cómo crear y usar estos resource-bundles en las primeras tres secciones. En la sección fina te enseñaremos cómo manejar tiempo, fechas, números, y monedas para diferentes locales.

Localización es todo acerca de hacer software relevante y usable para usuarios de diferentes culturas; en otras palabras, personalizar el software para diferentes países o lenguajes. ¿Cómo puedes localizar una aplicación? Dos importantes guías es necesario tomar en cuenta para localizar tu aplicación:

1. No hardcodear texto (como mensajes a los usuarios, elementos textuales en los GUIs, etc.) y separar estos en archivos externos o clases dedicadas. Con esto logrado se usa un mínimo de esfuerzo para añadir soporte para un nuevo locale en tu software.
2. Manejar aspectos culturales específicos como tiempo, fecha, moneda, y formatear números con localización en mente. En lugar de asumir un locale por default, el diseño en este locale es customizado.

NOTA: El texto no es la única cosa que necesita ser localizada en una aplicación. Por ejemplo, si tu aplicación usa audio, sonidos para dar instrucciones, estos deben cambiar según la localización. Similarmente, si el software muestra grafos o imágenes para un locale específico, estas deben ser transformadas.

# Locales

Un locale es un lugar que representa un país, lenguaje o cultura. Considera el locale Canada-French. El francés es hablado en muchas partes de Canada, y esto puede ser un locale. En otras palabras, si quieres vender software personalizado para Canadienses que hablan francés, entonces necesitas facilitar con tu software su locale. En Java, este locale es representado por el código fr\_CA; bueno discutiremos el esquema de nombres para locales en más detalle en esta sección.

## La clase Locale

En java, la clase Locale provee soporte para locales. La siguiente tabla muestra algunos métodos importantes en esta clase.

|  |  |
| --- | --- |
| static Locale[] getAvailableLocales() | Retorna una lista de locales disponibles (locales instalados) soportados por la JVM |
| static Locale getDefault() | Retorna el locale default de la JVM |
| static void setDefault(Locale newLocale) | Fija el locale default para la JVM |
| String getCountry() | Retorna el código del país del objeto locale |
| String getDisplayCountry() | Retorna el nombre del país del objeto locale |
| String getLangauge() | Retorna el código del lenguaje para el objeto locale |
| String getDisplayLanguage() | Retorna el nombre del lenguaje para el objeto locale |
| String getVariant() | Retorna el código del variante para el objeto locale |
| String getDisplayVariant() | Retorna el nombre del variante para el objeto locale |
| String toString() | Retorna un String compuesto por los códigos de los lenguajes locales, país, variantes, etc. |

El siguiente código detecta el locale default y checa los locales disponibles en la JVM.

[AvailableLocales.java](src/java/oca2/cap12/AvailableLocales.java)

Vamos a observar los métodos en el programa antes de analizar la salida del programa. Usas el método getDefault() en la clase Locale para obtener el código del locale por default. Después de esto usas el método getAvailableLocales() en la clase Locale para obtener una lista (array de locale) de los locale disponibles en tu JVM. Ahora, por cada locale imprimes el código del locale llamando implícitamente al método toString() del locale y también imprimes el nombre descriptivo usando el método getDisplayName() del locale.

Hay cuatro diferentes tipos de locale en la salida del programa:

1. Solo un código, como en uk (para Ucraniano).
2. Dos códigos separados por guion bajo, como en ms\_My, que está para Malaysa y MY para Malayo
3. Tres códigos separados por guion bajo, como en no\_NO\_NY donde no está para Norway, NO para Norwegian y NY para Nynorsk.
4. Dos o tres códigos separados por guion bajo y finalizando por # o \_# como th\_TH\_TH\_#u-nu-thai, que discutiremos ahora. Aquí está como estos nombres son codificados.

lenguaje + “\_” + país + “\_” + (variante + “\_#” | “#”) + script + “-” + extensiones

Para el locale th\_TH\_TH\_#u-nu-thai, es:

* El código del lenguaje es th (Thai) y siempre se escribe en minúsculas.
* El código del país es TH (Thailand) y siempre se escribe en mayúsculas.
* El nombre del variante es TH; aquí se repite el código del país, pero puede ser cualquier Srting.
* El nombre del script es una String vacía aquí; si se da, debe ser una String de cuatro letras con la primera en mayúsculas y las demás, minúsculas (ejemplo: Latin).
* La extensión es siguida pro # o \_#; esta es u-nu-thai en este ejemplo.

Este esquema de codificación es para permitir variaciones de programación incluso en el mismo lenguaje. Por ejemplo, el inglés es hablado en muchos países, y hay variaciones del lenguaje basadas en el país. Ahora sabemos que el inglés americano es diferente del inglés Británico, pero hay muchas versiones. Vamos a cambiar el loop en el código anterior para listar solo los locales que están relacionados al inglés.

[AvailableLocales2.java](src/java/oca2/cap12/AvailableLocales2.java)

La salida refiere a diferentes locales para inglés. Tu usas el método getLanguage() en el Locale, que retorna el código local. ¿Qué son los otros métodos? Ahora los vamos a explorar.

## Obtener detalles del Locale

Los métodos getter en la clase Locale como getLanguage(), getCoutry(), y getVariant() retornan códigos, mientras que los métodos similares getDisplayCountry(), getDisplayLanguage() y getDisplayVariant() retornan los nombres. El siguiente código muestra cómo usar estos métodos para el Locale.CANADA\_FRENCH.

[LocaleDetails.java](src/java/oca2/cap12/LocaleDetails.java)

Vamos a entender el programa. El método setDefault() toma un objeto Locale como argumento. En este programa fijas el default Locale como Locale.setDefault(Locale.CANADA\_FRENCH);

La clase Locale tiene muchos objetos Locale estáticos representando los Locale más comunes, por lo que no tienes que instanciarlos y los puedes usar en tu programa directamente. En este caso, Locale.CANADA\_FRENCH es un objeto Locale estático. En lugar de usar este objeto estático, puedes escoger instanciar un objeto Locale. Aquí este una manera alternativa de fijar el Locale por default creando un objeto Locale.

[LocaleDetails2.java](src/java/oca2/cap12/LocaleDetails2.java)

El método getDefault() en la clase Locale retorna el objeto Locale por default que está fijado para la JVM. La siguiente sentencia usa métodos para obtener información relacionada con el país. La diferencia entre getCountry() y getDisplayCountry() es que el primer método retorna el código del país (que no es muy legible para los humanos), y el segundo retorna el nombre del país, que es humanamente legible. El código del país es un código de dos o tres letras (Este código forma parte de un estándar internacional: ISO 3166).

El comportamiento de los métodos getLanguage() y getDisplayLanguage() es similar para obtener los detalles del país. El código de lenguaje consiste en dos o tres letras, y este código viene de otro estándar internacional: ISO 639.

Aquí no hay variante en este Locale, por lo que nada será impreso cuando uses el método getVariant() y getDisplayVariant(). Sin embargo, para cualquier otro locale, puede haber valores de variantes, y estos valores serán impresos para este locale. El variante puede ser cualquier cantidad de detalles extra como los entornos de operación (como MAC para una computadora Macintosh) o el nombre de una compañía (como Sun o Oracle).

Otro que esto, también tienes otros pocos métodos menos usados como getScript(), que retorna el código script para el locale.

# Resource Bundles

En la última sección, hemos discutido la clase Locale y la manera de obtener detalles del locale por default, así como obtener una lista con los Locale disponibles. ¿Cómo puedes usar esta información para personalizar tus programas? Vamos a tomar un ejemplo simple para saludar a alguien: en inglés dices “Hello” pero si el locale es diferente, ¿Cómo puedes cambiar este saludo, por ejemplo “Hola” si el locale es México y español?

Una solución obvia es obtener el locale por default, checar si el locale es México e imprimir “Hola”. Funcionará pero esta aproximación no es muy flexible ni extensible. ¿Cómo puedes personalizar para otros locales como Arabia Saudita (Arabic) o India (Hindi)? Puedes encontrar y reemplazar todas las strings locale para personalizar por cada locale; esta tarea puede ser una pesadilla si tu aplicación consiste en cientos de cadenas dispersadas en millones de líneas de código.

En java los resource-bundles proveen una solución para este problema de cómo customizar la aplicación para el locale específico. Entonces ¿Qué es un resource-bundle? Es un conjunto de clases o archivos de propiedades que te ayudan a definir un juego de claves y mapear estas llaves a valores locale específicos.

La clase abstracta ResourceBundle provee una abstracción de resource-bundles en Java. Tiene dos clases derivadas: PropertyResourceBundle y ListResourceBundle. Estas dos clases derivadas proveen soporte para los resource-bundles usando dos mecanismos diferentes.



**ResourceBundle**: provee soporte para múltiples locale; esta clase abstrae la funcionalidad de localización.

**ListResourceBundle**: Maneja recursos para un locale como una lista. Por cada locale, extiende esta clase y sobre-escribe el método getContents().

**PropertyResourceBundle**: Maneja los recursos para un locale como un archivo de propiedades. Usa el método ResourceBundle.getBundle() para cargar automáticamente el archivo de propiedades para el locale.

* **PropertyResourceBundle**: Esta clase concreta provee soporte para múltiples locale en forma de archivos de propiedades. Para cada locale, tu especificas las claves y los valores en un archivo de propiedades. Para un locale dado, si usas el método getBundle(), el archivo de propiedades adecuado será cargado automáticamente. Por supuesto, no hay magia en esto; tienes que seguir ciertas convenciones de nombres para crear archivos de propiedades, que discutiremos en la sección dedicada para esto. Puedes usar solo Strings como claves y valores cuando usas archivos de propiedades.
* **ListResourceBundle**: Para añadir soporte para un locale, puedes extender a esta clase abstracta. En tu clase derivada, tienes que sobre-escribir el método getContents(), que retorna un array bidimensional de la clase Object (Object[][]). Este array debe tener la lista de claves y valores. Las claves deben ser Strings. Típicamente los valores son también Strings, pero los valores pueden ser cualquiera: sonidos, videos, URLs, o imágenes.

Vamos a dar un vistaso rápido a los métodos soportados por la clase abstracta ResourceBundle. La siguiente tabla resume importantes métodos de esta clase. Ahora discutiremos la localización soportada usando estas dos clases derivadas de ResourceBundle.

|  |  |
| --- | --- |
| Método | Descripción corta |
| Object getObject(String key) | Retorna el valor mapeado a la clave dada. Lanza una MissingResourceException si no se encuentra objeto para la clave dada |
| static ResourceBundle getBundle(String baseName)  static ResourceBundle getBundle(String baseName, Locale locale)  final ResourceBundle getBundle(String baseName, Locale targetLocale, Control control) | Retorna el ResourceBundle del baseName, locale y control dados; lanza una MissingResourceException si no coincide ningún resource-bundle. La instancia Control está diseñada para controla u obtener información acerca del proceso de carga del resource bundle. |
| String getString(String key) | Retorna el valor mapeado a la clave dada; equivalente a castear el valor de getObject() a String. Lanza una MissingResourceException si no hay un objeto para la clave dada. Lanza una ClassCastException si el objeto retornado no es un String. |

## Usar PropertyResourceBundle

Si diseñas tu aplicación con localización en mente usando archivos de propiedades, puedes agregar soporte para locales nuevos a la aplicación sin cambiar nada en el código.

Vamos a ver un ejemplo usando archivos de recursos y será más claro para ti. Vamos a empezar con un programa simple que imprima “Hola” al usuario. Este programa tiene tres archivos de propiedades:

1. El resource bundle que asume que el español es el locale por default
2. Uno para Arabic
3. Uno para Italian

Como se discutió antes, los archivos de propiedades definen strings como pares clave-valor en un archivo. Aquí hay un ejemplo de un classpath que puede ser mapeado al actual path en tu máquina: classpath = C:\Program Files\Java\jre7.Property archivos que usualmente contienen numerosos valores clave-valor, con cada par separado en diferentes líneas como en el ejemplo siguiente:

[PropertiesExample.properties](src/java/oca2/cap12/PropertiesExample.properties)

Para el caso de localización, usas archivos de propiedades para mapear las mismas claves String a valores String. En el programa te referirás a las claves y al cargar la propiedad coincidente del archivo para el locale, el valor correspondiente para las claves será encontrado en el archivo.

El nombre de este archivo de propiedades es importante (lo verás pronto) y debajo está el contenido de estos bundles. Para mantener este ejemplo simple, solo hay un para clave-valor en este archivo de propiedades; en programas reales, puede haber cientos y hasta miles de pares representados en cada archivo de propiedades.

Como puedes ver, el bundle por default está llamado ResourceBundle.properties. El árabe es RersourceBundle\_ar.properties y similarmente el italiano es llamado ResourceBundle\_it.properties, que hacen uso del sufijo \_ar e \_it que se asocian con sus respectivos lenguajes. El siguiente código muestra cómo usar los estos archivos.

[LocalizedHola.java](src/java/oca2/cap12/LocalizedHola.java)

Hay dos opciones para cambiar el locale por default de la JVM

1. Llamando al método Locale.setDefault(Locale.ITALY)
2. Especificar el default locale cuando invocas a la JVM desde la línea de comandos.
   1. >java –Duser.language=it ClassName

Como puedes ver, dependiendo del locale puedes fijarlo explícitamente, dejar un locale por default, en este caso es\_MX; el archivo de propiedades correspondiente será cargado y el mensaje es resuelto.

NOTA: Si te olvidas de crear el archivo de propiedades o este no está en el path, obtendrás un MissingResourceException.

En el programa primero obtienes el locale actual en la sentencia:

Locale.lDefault = Locale.getDefault();

Después de esto cargas el resource-bundle que empieza con el nombre ResourceBundle y pasa el locale para el que se cargará el resource-bundle.

ResourceBundle rb = ResourceBundle(“ResourceBundle”, lDefault);

Finalmente, del resource-bundle, buscas la llave “Saludo” y usas el valor de esta basado en el resource-bundle cargado.

System.out.println(rb.getString(“Saludo”));

## Usar ListResourceBundle

Nota que las claves son Strings, pero los valores pueden ser de cualquier tipo, por lo tanto el tipo del array es Object; entonces, el método retorna una lista de pares clave-valor. Como un resultado, el método getContents() retorna un array bidimensional de Objects.

El siguiente código muestra un ejemplo de extender la clase abstracta ListResourceBundle, que se supone que retorna una caja de películas de oficina para un particular locale. Este define un resource-bundle llamado ResBundle. Ya que el nombre de la clase no tiene ningún sufijo (como \_it o \_en\_US), esta es la implementación por default de este resource-bundle. Cuando buscas para encontrar un ResBundle para cualquier locale, esta implementación por default será usada en caso de no encontrarse la otra.

[ResBundle.java](src/java/oca2/cap12/ResBundle.java)

Ahora vamos a definir un ResBundle para el locale italiano. Debes de darle el sufijo a la clase \_it\_IT. El código del lenguaje está para el país Italia.

[ResBundle\_it\_IT.java](src/java/oca2/cap12/ResBundle_it_IT.java)

Como puedes ver, la implementación para ResBundle y ResBundle\_it\_IT es similar a excepción de los valores mapeados a las claves. Ahora ¿Cómo sabes si los resource-bundles están trabajando o no? El siguiente código carga ambos bundles.

[LocalizedHola2.java](src/java/oca2/cap12/LocalizedHola2.java)

Carga el default y el italiano exitosamente. Sin embargo, hay problemas con esta salida. El valor 2782275172 está en dólares y el valor 43000000 está en euros. Más que los números se imprimen sin comas, por lo que es difícil leerlos. Estos valores necesitan ser localizados también. Regresaremos a ver este tema en la última sección de este capítulo.

Ahora considera la siguiente sentencia:

Long revenue = (Long)(resBundle.getObject(“GrossRevenue”));

Esta sentencia retorna el valor mapeado a la llave GrossRevenue en el resource-bundle. Lo tienes definido como como un objeto entero en las clases las dos clases ResBundle y funcionan. Si casteas los tipos incorrectamente obtendrás un ClassCastException.

Además, nota el nombre de la clave es sensible a mayúsculas y minúsculas. Si te equivocas en el nombre o en el case entonces obtendrás un MissingResourceException.

NOTA: Has creado resource-bundles extendiendo a la clase abstracta ListResourceBundle, mientras que con PropertyResourceBundle necesitas archivos de propiedades. Por lo tanto, extendiendo a ListResourceBundle puedes tener cualquier tipo de objetos como valores, mientras que en las filas de propiedades solo pueden ser Strings.

# Cargar un resource-bundle

Ya has cargado resource-bundles en el programa que es escrito usando ResourceBundle o sus dos clases derivadas. Necesitas entender el proceso de esta carga a fondo, lo que será cubierto en esta sección.

El proceso para encontrar los recursos es el mismo para las dos clases que extienden de la clase abstracta base ResourceBundle.

NOTA: Para los resource-bundles implementados como clases que extienden ListResourceBundles, Java usa el mecanismo de reflexión para encontrar y cargar las clases. Necesitas asegurarte que la clase es pública para que el mecanismo de reflexión pueda encontrar la clase.

## Convenciones de nombres para los resource-bundles

Java refuerza una convención de nombres predefinida a seguir, para crear resource-bundles. Solo a través de los nombres de los archivos java cargará el archivo correcto. Por lo tanto es importante entender esta convención cuando creas resource-bundles para localizar aplicaciones Java.

Ya has visto cómo un nombre locale es codificado. Entender este nombre es importante para nombrar a los resource-bundles porque este hace uso del mismo esquema de codificación. Un resource-bundle cualificado tiene la siguiente forma:

packagequalifier.bundlename\_language\_country\_(variant + “\_#” | “#”)script-extencions

Aquí está la descripción de cada uno de los elementos en el nombre completamente cualificado:

1. **packagequalifier**: el nombre del paquete (o sub-paquetes) en donde está provisto el resource-bundle.
2. **bundlename**: el nombre del resource-bundle que usas en el programa para referirlo y cargarlo.
3. **language**: una abreviación de dos letras típicamente dadas en minúsculas para el lenguaje del locale (en raras veces también puede contener tres letras).
4. **country**: Una abreviación de dos letras típicamente dadas en mayúsculas para el país del locale (en raros casos puede contener tres letras).
5. **variant**: Una lista arbitraria de variantes (en mayúsculas o minúsculas) para diferenciar locales cuando tienes más de un locale para una combinación de lenguaje y país.

Omitiremos describir la extensión script ya que es raramente usada.

Por ejemplo, considera este nombre cualificado completamente:

localization.examples.AppBundle\_en\_US\_Oracle\_exam

En este caso, localization.examples es el paquete, AppBundle es el nombre del resource-bundle, en es el lenguaje (inglés), US es el país, y Oracle\_exam es la variante.

Las dos (a veces tres) abreviaciones de letras para el lenguaje y el país son predefinidas ya que son basadas en estándares internacionales. Nosotros no proveemos la lista detallada y no hay necesidad de saber o recordar todas ellas. Puedes buscar en la documentación de la clase Locale para entender esto.

NOTA: En el examen no esperan que memorices los códigos de lenguaje o país que son usados para nombrar resource-bundles. Sin embargo, se espera que recuerdes la convención de nombres y reconozcas los elementos que constituyen un nombre cualificado completamente.

Dado esto puede haber muchos resource-bundles para un nombre de bundle, ¿Cuál es la secuencia para determinar el resource-bundle que será cargado? Para clarificar, te presentamos la secuencia como una serie de pasos. La búsqueda empieza desde el paso 1. Si en algún paso se encuentra una coincidencia, el resource-bundle será cargado. De lo contrario la búsqueda continuará en el siguiente paso.

* **Paso 1**: La búsqueda empieza por encontrar una coincidencia exacta para el resource-bundle con el nombre completo.
* **Paso 2**: El último componente (la parte separada por \_) es desechada y la búsqueda es repetida con el nombre más corto resultante. Este proceso es repetido hasta que el último elemento es dejado.
* **Paso 3**: La búsqueda vuelve a empezar usando el nombre completo del bundle para el locale por default.
* **Paso 4**: Busca el resource-bundle con solo el nombre del bundle.
* **Paso 5**: La búsqueda falla y lanza un MissingBundleException (no-checada).

Aquí un ejemplo

* BundleName\_language\_country\_variant
* BundleName\_language\_country
* BundleName\_language
* BundleName\_defaultLanguage\_defaultCountry
* BundleName\_defautlLanguage

Considera un ejemplo para entender cómo el resource-bundle buscado es encontrado, y esto te será más claro. Asume que tienes las siguientes cinco entradas en el path de búsqueda, y tu default locale es US English.

* ResourceBundle.properties -- Global
* ResourceBundle\_ar.properties -- Lenguaje arábico
* ResourceBundle\_en.properties -- Inglés (asumiendo que en\_US es el default)
* ResourceBundle\_it.properties -- Italiano
* ResourceBundle\_it\_IT\_Rome.properties – Italiano (Italia,Roma,Vaticano)

NOTA: el método getBundle() toma un objeto ResourceBundle.Control como parámetro adicional. Extendiendo a esta clase ResourceBundle.Control y pasando la instancia de esta clase extendida al método getBundle() puedes cambiar el proceso de búsqueda de bundles por default, o leer de formatos no-estándar como XML.

Entonces tu extiendes este ResourceBundle.Control y sobre-escribes el método getCandidateLocales(). Esto es para listar los candidatos de ocales programáticamente y finalmente mostrar el locale que coincide. El siguiente programa muestra cómo hacer esto.

[CandidateLocales.java](src/java/oca2/cap12/CandidateLocales.java)

Ahora, antes de intentar con otros locales, considera cómo el programa trabaja. Para trazar cómo Java resuelve el resource-bundle es finalmente cargado, necesitas obtener la lista de candidatos locale. Con el método ResourceBundle.getBundle() puedes pasar un argumento adicional que sea instancia de ResourceBundle.Control. Por esto defines la clase TalkativeResourceBundleControl.

La clase TalkativeResourceBunldeControl extiende a la clase ResourceBundle.Control y sobre-escribe el método getCandidateLocales(). Este método retorna una List<Locale> que contiene la lista de candidatos para el locale dado. Invocas a super.getCandidateLocales() y atraviesas la List resultante para imprimir los candidatos para que puedas examinar la salida después. Para este método sobre-escrito simplemente retornas la List<Locale>. Entonces el comportamiento de la clase TalkativeResourceBundleControl es idéntica que ResourceBundle.Control, con excepción que el método sobre-escrito imprime los candidatos.

La clase CandidateLocales hace uso de TalkativeResourceBundleControl. Esta tiene un métdo helper llamado loadResourceBundle() que toma el nombre del resource-bundle y el nombre de los argumentos de locale. Este método simplemente pasa estos valores de argumentos al método ResourceBundle.getBundle(); adicionalmente instancias a TalkativeResourceBundleControl y pasa este objeto como tercer argumento del método. El método getBundle() retorna un objeto ResourceBundle. Si el nombre del locale de ResourceBundle.getLocale() está vacío, significa que no java ha cargado el resource-bundle global. (Recuerda que el resource-bundle global para este nombre de bundle, no tiene detalles asociados de locale). Si el nombre del locale no está vacío, significa que java ha resuelto a un locale particular.

Ahora considera el código en el método main(). Este llama al método loadResourceBundle() para el locale it\_IT\_Rome. Aquí, hay tres candidatos locale que pueden coincidir con el archivo de propiedades correcto it\_IT\_Rome. Entonces tu sabes que cargará el archivo de propiedades ResourceBundle\_it\_IT\_Rome.properties correctamente.

Para continuar este experimento, vamos a cambiar el código dentro del método main() del programa anterior, instanciando un loclale con los argumentos new Locale(“fr”, “CA”) y observa qué pasa.

Como el archivo de propiedades correspondiente a fr\_CA no existe, entonces el método sigue buscando el archivo de propiedades por default, como este último sí se encuentra, el método resuelve y carga el archivo de propiedades por default.

# Formatear para la cultura local

El texto es obviamente el aspecto principal que hay que localizar. Sin embargo, hay muchos aspectos que son manejados de manera diferente, basados en su locale: fecha, hora, números, moneda. Bueno, discutiremos cada tema en detalle ahora.

Para localizar texto, la mejor aproximación que necesitas seguir es no harcodear los Strings. La idea clave para recordar para fecha, tiempo, moneda y números es usar el formateo de acuerdo a la cultura para localizarlo. La siguiente figura muestra cómo dos importantes clases que discutiremos en esta sección, NumberFormat y DateFormat, son ambas heredadas de la clase base Format; estas clases son parte del paquete java.text y son útiles para localización de software.



* **Format**: Clase abstracta base que soporta el formateo sensible a la localidad como fechas, tiempo y moneda.
* **NumberFormat**: Clase base para procesar números por porcentaje, moneda, etc.
* **DateFormat**: Clase base para procesar fechas y tiempo basado en el locale
* **SimpleDateFormat**: Te permite definir un patrón personalizado para procesar fechas

## La clase NumberFormat

Esta clase provee soporte para procesar números de manera locale. Por ejemplo, dependiendo del locale, cómo cientos son separados, los caracteres de puntuación usados como separadores, imprimir una cantidad como un valor de moneda, etc., son diferentes y esta clase provee esta funcionalidad.

La clase NumberFormat provee métodos para formatear o analizar números. Aquí “formatear” significa convertir un valor numérico a un String de acuerdo a la cultura; similarmente, “analizar” significa convertir el número de vuelta a su forma numérica. Por ejemplo, si quieres imprimir el valor constante long 10\_000\_000L en diez millones en Alemania, tienes que formatear este valor pasando al método format() en la clase NumberFormat, y este método retornará un String “10.000.000” (nota que se usan puntos para separar los caracteres en miles). Ahora, si tu lees el valor 10 millones en Alemania para convertir este valor a un valor long y utilizarlo en el programa, puedes pasar el String al método parse(). El siguiente código muestra paso por paso cómo hacer esto.

[FormatNumber.java](src/java/oca2/cap12/FormatNumber.java)

Como puedes ver, el valor 10 millones es impreso en el formato alemán 10.000.000. Para parsearlo y regresarlo a long, usas el método parse() de NumberFormat, que retorna un Number si el parseo es exitoso, de lo contrario lanzará una excepción ParseException.

Nota que el método parse() es diferente del método format. El método parse() lee números provistos en String e intenta convertirlos a números. El método format() es usado para imprimir valores de acuerdo a los valores seteados en el objeto NumberFormat. El siguiente código muestra la diferencia entre ambos.

[FractionDigits.java](src/java/oca2/cap12/FractionDigits.java)

El método parse() lee los valores y los convierte a Number si es satisfactorio. Entonces, este no usa el máximo número de fracciones establecido con el método setMaximumFractionDigits(); sin embargo, si usas el método format, este sí usa esta característica, y así se explican las diferencias de ambos métodos en la salida del programa.

En la siguiente tabla se muestran importantes métodos de la clase NumberFormat. Los métodos estáticos que empiezan con el prefijo “get” y terminan con el sufijo “instance”, como getCurrencyInstance(), son métodos Factory soportados por esta clase.

|  |  |
| --- | --- |
| Método | Descripción corta |
| String format(double number)  String format(long number) | Formatea el número de acuerdo al locale del NumberFormat. Los primeros dos métodos sobrecargados usan un StringBuffer implícito, mientras que los dos últimos lo usan explícito para construir el String. |
| Number parse(String source) | Parsea el número del String dado. Retorna un Long o Double dependiendo del valor que dé el número en el origen. Lanza un ParseException si el parseo falla. |
| static Locale[] getAvailableLocales() | Retorna una lista de los locale soportados por la JRE para formateo de números |
| static NumberFormat getInstance() | Método Factory que retorna un objeto de la clase NumberFormat del locale por default. |
| Currency getCurrency() | Retorna una instancia Currency usada por el objeto NumberFormat. |
| static Currency getCurrencyInstance() | Retorna la instancia de NumberFormat adecuada para formato de moneda; una versión overloaded de este método toma un Locale como argumento. |
| static NumberFormat getIntegerInstance() | Retorna una instancia de NumberFormat adecuada para formatear números enteros; una versión sobrecargada toma un Locale como argumento. |
| static NumberFormat getPercentInstance() | Retorna una instancia de NumberFormat adecuada para formatear porcentajes; una versión sobrecargada de este método, toma un Locale como argumento. |

La clase NumberFormat soporta la impresión de valores de moneda. Puedes usar el método getCurrencyInstance(), que retorna un objeto Currency. El siguiente código ilustra el uso de estos métodos para imprimir el valor 10 millones en cuatro diferentes locale (sin ejecutar tipo de la conversión de tipo de cambio).

[LocalizedCurrency.java](src/java/oca2/cap12/LocalizedCurrency.java)

Como puedes ver usando el objeto NumberFormat retornado por el método getCurrencyInstance (locale), puedes formatear números para imprimirlos como valores de moneda para el locale. También puedes usar la clase Currency independientemente de la clase NumberFormat, como discutiremos ahora.

## La clase Currency

La siguiente tabla lista importantes métodos en la clase Currency.

|  |  |
| --- | --- |
| Método | Descripción corta |
| int getNumericCode() | Retorna el código numérico ISO 4217 del Currency |

|  |  |
| --- | --- |
| Método | Descripción corta |
| int getDefaultFractionDigits() | Retorna el número de dígitos usados por default en este Currency, como 0 para el Yen japonés y 2 para US dollar. |
| String getDisplayName() | Retorna la descripción legible del Currency para el locale subyacente, por ejemplo US Dollar |
| String getDisplayName(Locale locale) | Retorna la descripción legible del Currency para el locale dado. |
| static Currency getInstance(String currencyCode) | Retorna un objeto Currency correspondiente al código currency dado. |
| static Currency getInstance(Locale locale) | Retorna el objeto Currency correspondiente al locale dado |
| static Set<Currency> getAvailableCurrencies() | Obtiene la lista de instancias Currency disponibles en el JDK. |
| String getSimbol(Locale locale) | Retorna el símbolo de moneda para el locale dado |
| String getCurrencyCode() | Retorna el código de moneda (ISO 4217) para el locale de la instancia Currency |

El siguiente código muestra cómo hacer uso de algunos de estos métodos.

[CurrencyDetails.java](src/java/oca2/cap12/CurrencyDetails.java)

La salida se explica por sí sola. Nota que para muchos locale que no hay un símbolo, el método getSymbol() solo retorna el código de la moneda.

## La clase DateFormat

La clase DateFormat provee soporte para procesar fechas y tiempo en el locale. La siguiente tabla muestra algunos de los métodos importantes en esta clase.

|  |  |
| --- | --- |
| Método | Descripción corta |
| String format(Date date) | Formatea la fecha dada para el locale por default y retorna una representación textual. Su versión sobrecargada toma un StringBuffer y la posición como argumentos y retorna un objeto StringBuffer; útil si un StringBuffer necesita ser formateado como fecha |
| Date parse(String source) | Lee la String dada de acuerdo a su locale para retornar un objeto Date; lanza una excepción de tipo ParseException si falla. Hay una versión sobrecargada que toma la posición desde donde se tiene que parsear el String, como argumento adicional |
| static Locale[] getAvailableLocales() | Retorna un array de Locale soportados por JR para formateo de tiempo y fechas |
| static DateFormat getInstace() | Retorna el DateFormat por default. Usa DateFormat.SHORT para fecha y hora |
| static DateFormat getDateInstance() | Retorna una instancia para procesar fechas; por default locale. Dos versiones sobrecargadas toman un style y Locale como argumentos adicionales |
| static DateFormat getTimeInstance() | Retorna una instancia para procesar tiempo; por default locale. Dos versiones sobrecargadas toman style y Locale como argumentos adicionales |
| static DateFormat getDateTimeInstance() | Retorna una instancia para procesar fecha y hora por default locale. Dos versiones sobrecargadas toman style y Locale como argumentos adicionales |

Dependiendo del locale, la fecha o tiempo impresos pueden ser considerablemente diferentes, como se muestra la salida del siguiente programa.

[DatePrint.java](src/java/oca2/cap12/DatePrint.java)

Este programa obtiene una instancia de DateFormat usando una de las versiones sobrecargadas del método getDateInstance(). Este método toma un estilo de formato de impresión como primer argumento, y el locale para el que será usado el formato como segundo argumento. ¿Qué es esto de los estilos de formato? El siguiente programa muestra los cuatro estilos y cómo se ven las fechas para cada uno de estos estilos.

[DateStyleFormat.java](src/java/oca2/cap12/DateStyleFormat.java)

Como puedes, puedes obtener una instancia de DateFormat para un estilo predefinido, basado en las necesidades. El estilo por default es DateFormat.MEDIUM.

El DateFormat tiene tres métodos Factory sobrecargados: getDateInstance(), getTimeInstance() y getDateTimeInstance(), que retornan instancias DateFormat para procesar fecha, tiempo o ambos. El siguiente código muestra cómo usarlos.

[DateTimePrint.java](src/java/oca2/cap12/DateTimePrint.java)

Este programa muestra cómo obtener instancias de DateFormat para procesar fecha, tiempo y ambos. Si quieres crear tu propio formato o patrón para procesar la fecha o el tiempo, necesitas utilizar la clase SimpleDateFormat.

## Clase SimpleDateFormat

Esta clase extiende a DateFormat. Esta clase usa el concepto de patrón para formatear el tiempo y la fecha. Antes debes recordar la creación de string-patterns. Vamos a dar un primer vistaso para aprender cómo personalizar nuestro propio formato para imprimir fecha y tiempo.

[PatternStringExample.java](src/java/oca2/cap12/PatternStringExample.java)

En este programa codificas el formato de la fecha o tiempo usando letras de un patrón de fecha o tiempo. Usualmente estas letras son repetidas en el patrón. Nota que las mayúsculas o minúsculas significan cosas diferentes, por lo que tienes que leer la codificación cuidadosamente cuando intentes usar estas letras. Por ejemplo, en dd-MM-yy, MM se refiere al mes; sin embargo en dd-mm-yy, mm se refiere a minutos.

En este programa, diste un simple ejemplo para crear un formato personalizado de fecha. Letras similares están disponibles para crear patrones personalizados de fecha y hora. Aquí está una lista de letras importantes y su significado para crear patrones de datos.

* **G** – era (AC/DC)
* **y** – año
* **Y** – semana del año
* **M** – Mes (en año)
* **w** – semana (en año) cuenta las semanas en el año
* **W** – semana (en mes)
* **D** – día (en año)
* **d** – día (en mes)
* **F** – día de la semana en mes
* **E** – nombre de día de la semana
* **u** – número de día de la semana (rango del 1 al 7)

El siguiente programa usa esto para dificultar el patrón para crear formatos personalizados de fecha.

[CustomDatePatterns.java](src/java/oca2/cap12/CustomDatePatterns.java)

Como puedes ver, repetir letras resulta en un resultado más largo. Por ejemplo, cuando usas E (que es el nombre de la semana), este imprime Miércoles, mientras que EEEE imprime el nombre completo del día de la semana. Si das un patrón incorrecto u olvidas usar comillas simples para separar tu texto del patrón, obtendrás un IllegalArgumentException por pasar un patrón ilegal.

Ahora, observa un ejemplo similar para crear patrones personalizados para la hora. Aquí está la lista de letras importantes.

* **a** – Marca para el texto am/pm
* **H** – Hora (0-23)
* **k** – Hora (1-24)
* **K** – Hora en am/pm (0-11)
* **h** – Hora en am/pm (1-12)
* **m** – Minuto
* **s** – Segundo
* **S** – Milisegundo
* **z** – Zona horaria (formato general de zona horaria)

Para más letras y su especificación, analiza el javaDoc para SimpleDateFormat. El siguiente código es un programa que usa patrones para crear formatos de hora personalizados.

[CustomTimePatterns.java](src/java/oca2/cap12/CustomTimePatterns.java)

# Puntos a recordar

* Hay muchas maneras para obtener o crear un objeto Locale. Listamos cuatro opciones aquí para crear una instancia del Locale italiano.
  + **Opción 1**: Usar el constructor de la clase Locale(String language, String country, String variant):
    - Locale locale1 = new Locale(“it”, “”, “”);
  + **Opción 2**: Usar el método de la clase Locale, forLanguageTag(String langaugeTag):
    - Locale locale2 = Locale.forLanguageTag(“it”);
  + **Opcion 3**: Construir un objeto Locale instanciando Locale.Builder y después llamar al método setLanguageTag() de este objeto:
    - Locale locale3 = new LocaleBuilder().setLanguageTag(“it”).build();
  + **Opción 4**: Usar las constantes finales predefinidas en la clase Locale:
    - Locale locale4 = new Locale.ITALIAN;

Puedes escoger la manera de crear un objeto Locale basado en tus necesidades. Por ejemplo, la clase Locale tiene solo unas pocas constantes definidas para locales. Si quieres un Locale de alguna de estas, puedes directamente usarlas, de otra manera puedes checar cual otra opción usar.

* En lugar de llamar el método getDisplayCountry() de la clase Locale, sin argumentos, puedes escoger la versión sobrecargada de getDisplayCountry(Locale), que toma un Locale como argumento. Esto imprimirá el nombre del país del Locale pasado. Por ejemplo, para llamar Locale.GERMANY.getDisplayCountry, obtienes la salida Deutschland (que es la manera alemana de referirse a este país); sin embargo, para el caso de Locale.GERMAY.getDisplayCountry(Locale.ENGLISH), obtienes la salida Germany (que es la manera ingles de referirse a este país).
* El método setMaximumFractionDigits(int n) no aplica para el método parse, solo para format.

# Resumen

Leer y ajustar un Locale usando el objeto Locale

* Un Locale representa el lenguaje, cultura o país; la clase Locale en Java provee una abstracción para este concepto.
* Cada Locale puede tener tres entradas: el lenguaje, el país y los variantes. Puedes usar los códigos estándar disponibles para lenguaje y país para formar etiquetas Locale. No hay tags estándar para variantesM puedes porveer Strings de variantes según tus necesidades.
* Los métodos getter en la clase Locale como getLanguage(), getCountry y getVariant() retornan códigos mientras que los métodos similares Display, retornan nombres
* El método getDefault() en Locale retorna el Locale por default ajustado para la JVM. Puedes cambiar este Locale default por otro usando el método setDefault.
* Hay muchas formas de crear u obtener un Locale correspondiente al locale:
  + Usar el constructor de la clase Locale,
  + Usar el método forLanguageTag(String langaugeTag), de la clase Locale,
  + Construir un Loclae instanciando Locale.Builder y después llamar al método setLanguageTag() de ese objeto,
  + Usar las constantes predefinidas finales para Locales en la clase Locale.

Construir un resource-bundle para cada Locale

* Un resource-bundle es un conjunto de clases o archivos de propiedades que ayudan a definir claves y mapearlas a los valores locales específicos.
* La clase ResourceBundle tiene dos clases derivadas: PropertyResourceBundle y ListResourceBundle. Puedes usar el método ResourceBundle.getBundle() para cargar automáticamente un resource-bundle para el Locale dado.
* La clase PropertyResourceBundle provee soporte para múltiples locale en la forma de archivos de propiedades. Para cada locale, puedes especificar las claves y valores en un archivo de propiedades. Puedes usar solo Strings como claves y valores.
* Para agregar soporte para un nuevo Locale, puedes extender la clase ListResourceBundle. En esta derivada clase puedes sobre-escribir el método Object[][] getContents(). El array retornado debe tener la lista de claves y valores. Las claves deben ser Strings y los valores pueden ser cualquier objeto.
* Cuando pasas la clave al método getObject() para encontrar el valor buscado en el resource-bundle, asegúrate que las claves pasadas y la llave en el resource-bundle, coincidan exactamente (el nombre de la clave es sensible a mayúsculas y minúsculas). Si estas no coinciden obtendrás un MissingResourceException.
* La conveción de nombres para el resource-bundle completo y cualificado es:
  + paquete.nombre\_lenguaje\_pais\_(variante + “\_#”| “#”)script-extensiones.

Cargar un resource-bundle en una aplicación

* El proceso para encontrar un resource-bundle coincidenet es la misma para archivos de propiedades y clases ListResourceBundle
* Aquí está la secuencia de búsqueda para encontrar un resource-bundle. Buscar empieza por el paso uno. Si en algún paso se encuentra el resource-bundle, este es cargado. De otra manera, la búsqueda procede al siguiente paso:
  + **Paso 1**: La búsqueda empieza por encontrar la coincidencia exacta del resource-bundle, por su nombre completo.
  + **Paso 2**: El último componente (la parte separada por \_) es eliminado y la búsqueda se repite con el nombre resultante. Este proceso es repetido hasta el último modificador quede.
  + **Paso 3**: La búsqueda reinicia usando el nombre completo del resource-bundle por default locale.
  + **Paso 4**: Se busca el resource-bundle con el nombre solo dado del bundle.
  + **Paso 5**: La búsqueda falla y se lanza un MissingBundleException.
* El método getBundle() toma un objeto ResourceBundle.Control como parámetro adicional al que puedes extender para poder controlar o personalizar la búsqueda del recurso y su carga en la memoria.

Formatear texto para Localizacion usando NumberFormat y DateFormat

* Para manejar fecha y hora, números y moneda de una manera sensible a lo local, puedes usar la clase java.text.Format y sus dos clases derivadas principales: NumberFormat y DateFormat.
* La clase NumberFormat provee soporte de acuerdo al Locale, para el manejo de números, en relación de cómo los miles son separados o tratando al número como un valor de moneda.
* La clase NumberFormat provee métodos para formatear o parsear números. “Formatear” significa convertir a valores numéricos a su manera textual para adecuarse a la vista de los usuarios; “Parsear” significa convertir un número a su forma numérica. El método parse() retorna un objeto Number si tiene éxito, de otra manera lanzará una ParseException.
* NumberFormat tiene varios métodos Factory: getInstance(), getCurrencyInstance(), getIntegerInstance() y getPercentInstance().
* La clase Currency provee soporte para manejar valores monetarios en un modo Locale.
* La clase DateFormat provee soporte para procesar fecha y hora de una manera localizada.
* La clase DAteFormat tiene tres métodos Factory sobrecargados: getDateInstance(), getTimeInstance y getDateTimeInstance() que retornan instancias de DateFormat para procesar fecha, hora y ambas.
* SimpleDateFormat (derivada de DateFormat) usa el concepto de patrón para soportar formatos personalizados para fecha y hora.
* Codificas el formato de la fecha u hora usando letras para formar un patrón.