Chapter 30

Localization

Localization

Localization is all about making the software relevant and usable for the users from different cultures– in other words, customizing software for people from different countries or languages.

Localization is often abbreviated as *l10n*

How do you localize a software application?

Two important guidelines should be heeded when you localize a software application:

- Do not hardcode text (such as messages to the users, textual elements in GUIs, etc.) and separate them into external files or dedicated classes. With this accomplished there is usually minimal effort to add support for a new locale in your software.

- Handle cultural-specific aspects such as date, time, currency, and formatting numbers with localization in mind. Instead of assuming a default locale, design in such a way that the current locale is fetched and customized.

In Java, it all starts with one class, **java.util.Locale.**

Locale Class

```
Locale locale = Locale.getDefault();
```

. Read and set the locale by using the Locale object

A locale is “a place representing a country, language, or culture.”

Locale representation

th_TH

1.The **language** part is required and it is always written in lowercase

The language code consists of two or three letters (this code comes from another international standard: ISO 639).

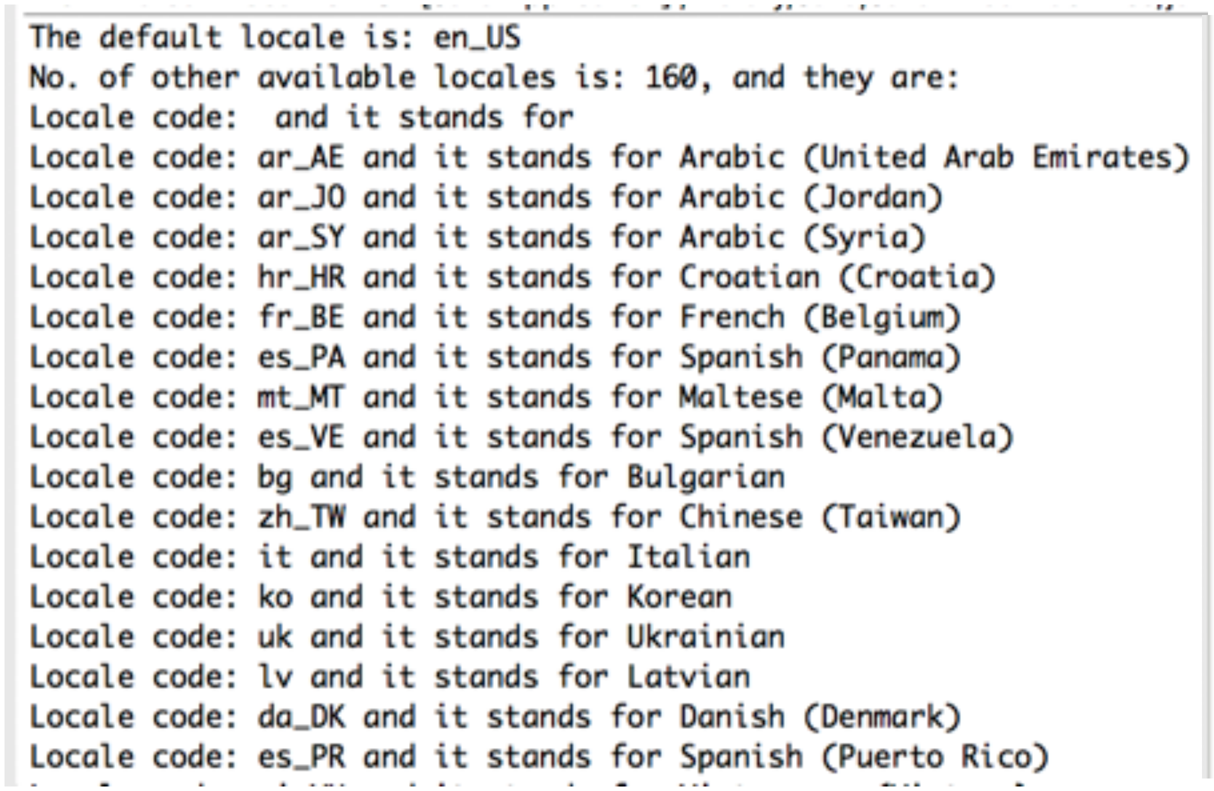
2.The **country** part is optional and it is always written in uppercase

The country code is a two or three letter code (this code comes from an international standard: ISO 3166).

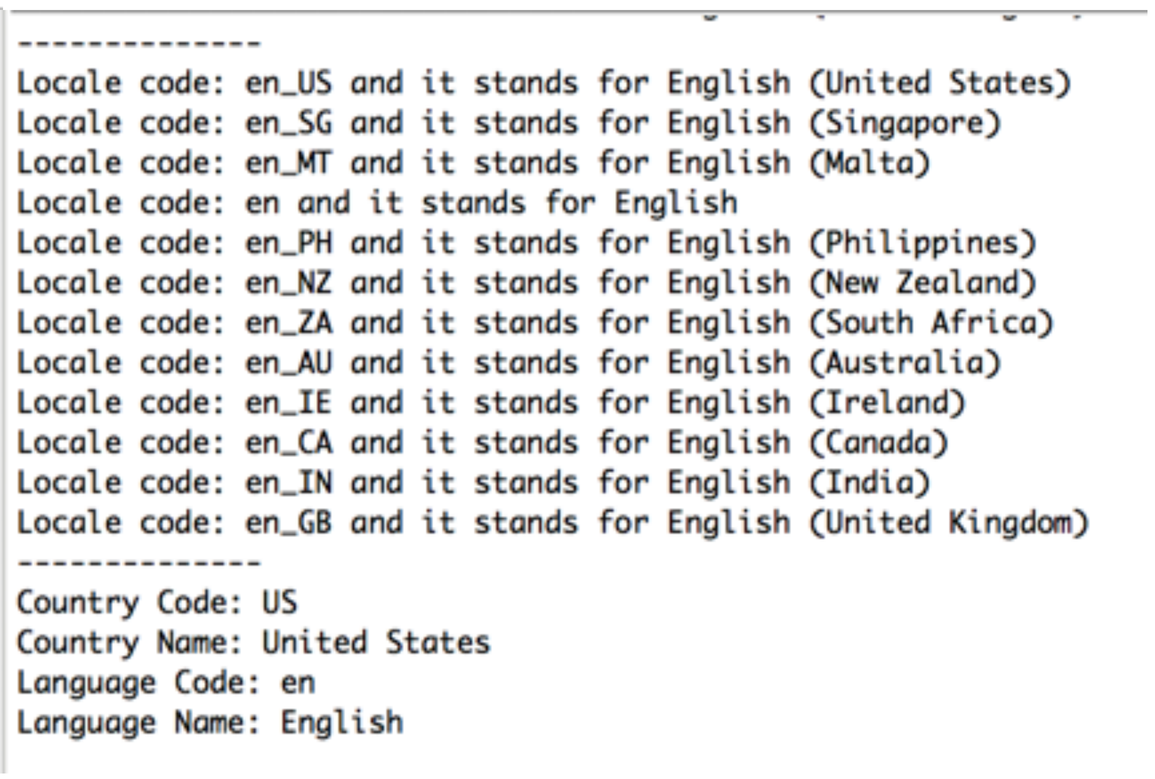
Notice the underscore for separation

Method	**Short Description**
`static Locale[] getAvailableLocales()`	Returns a list of available locales (i.e., installed locales) supported by the JVM.
`static Locale getDefault()`	Returns the default locale of the JVM.
`static void setDefault(Locale newLocale)`	Sets the default locale of the JVM.
`String getCountry()`	Returns the country *code* for the locale object.
`String getDisplayCountry()`	Returns the country *name* for the locale object.
`String getLanguage()`	Returns the language *code* for the locale object.
`String getDisplayLanguage()`	Returns the language *name* for the locale object.
`String getVariant()`	Returns the variant *code* for the locale object.
`String getDisplayVariant()`	Returns the *name* of the variant code for the locale object.
`String toString()`	Returns a `String` composed of the codes for the locale's language, country, variant, etc.

```java
public static void main(String[] args) {
     Locale locale = Locale.getDefault();
     System.out.println("The default locale is: " + Locale.getDefault());
     Locale [] locales = Locale.getAvailableLocales();
     System.out.printf("No. of other available locales is: %d, and they are: %n",
              locales.length);
     Arrays.stream(locales).forEach(loc -> System.out.printf("Locale code: %s and it stands for %s %n", loc, loc.getDisplayName()));
     System.out.println("--------------");
     Arrays.stream(Locale.getAvailableLocales())
     .filter(loc -> loc.getLanguage().equals("en"))
     .forEach(loc ->
              System.out.printf("Locale code: %s and it stands for %s %n",
                   loc, loc.getDisplayName()));
     System.out.println("--------------");
     System.out.println("Country Code: "+ locale.getCountry());
     System.out.println("Country Name: " + locale.getDisplayCountry());
     System.out.println("Language Code: " + locale.getLanguage());
     System.out.println("Language Name: " + locale.getDisplayLanguage());

}
```



```
The default locale is: en_US
No. of other available locales is: 160, and they are:
Locale code:  and it stands for
Locale code: ar_AE and it stands for Arabic (United Arab Emirates)
Locale code: ar_JO and it stands for Arabic (Jordan)
Locale code: ar_SY and it stands for Arabic (Syria)
Locale code: hr_HR and it stands for Croatian (Croatia)
Locale code: fr_BE and it stands for French (Belgium)
Locale code: es_PA and it stands for Spanish (Panama)
Locale code: mt_MT and it stands for Maltese (Malta)
Locale code: es_VE and it stands for Spanish (Venezuela)
Locale code: bg and it stands for Bulgarian
Locale code: zh_TW and it stands for Chinese (Taiwan)
Locale code: it and it stands for Italian
Locale code: ko and it stands for Korean
Locale code: uk and it stands for Ukrainian
Locale code: lv and it stands for Latvian
Locale code: da_DK and it stands for Danish (Denmark)
Locale code: es_PR and it stands for Spanish (Puerto Rico)
```



```
--------------
Locale code: en_US and it stands for English (United States)
Locale code: en_SG and it stands for English (Singapore)
Locale code: en_MT and it stands for English (Malta)
Locale code: en and it stands for English
Locale code: en_PH and it stands for English (Philippines)
Locale code: en_NZ and it stands for English (New Zealand)
Locale code: en_ZA and it stands for English (South Africa)
Locale code: en_AU and it stands for English (Australia)
Locale code: en_IE and it stands for English (Ireland)
Locale code: en_CA and it stands for English (Canada)
Locale code: en_IN and it stands for English (India)
Locale code: en_GB and it stands for English (United Kingdom)
--------------
Country Code: US
Country Name: United States
Language Code: en
Language Name: English
```

Setting Locales

1. Using a constructor There are three constructors:

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

For example:

```
Locale chinese = new Locale("zh");
Locale CHINA = new Locale("zh", "CN");
```

2. Using the **forLanguageTag(String)** factory method
This method expects a language code, for example:

```
Locale german = Locale.forLanguageTag("de");
```

3. Using Locale.Builder
You can set the properties you need and **build** the object at the end, for example:

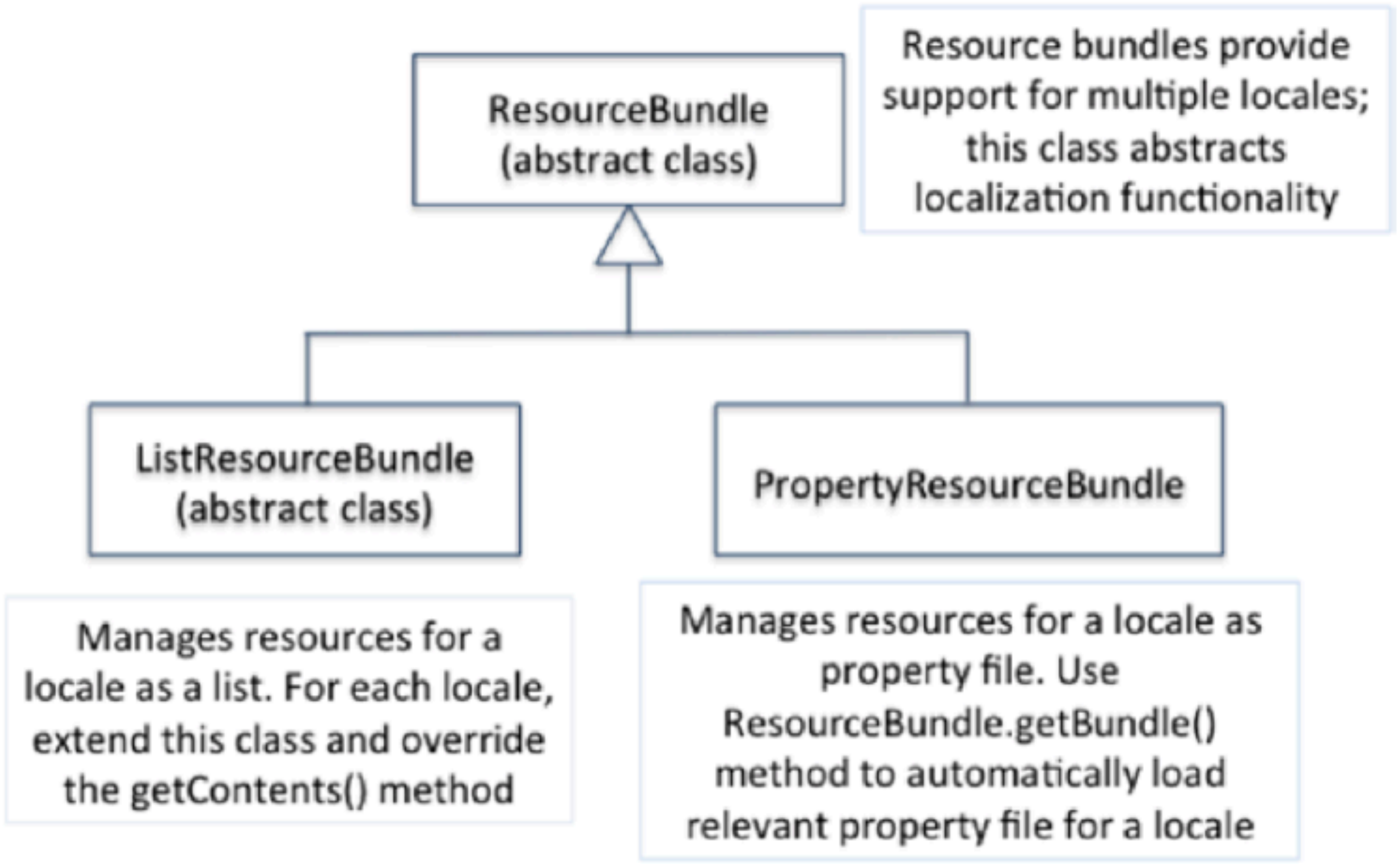
```
Locale japan = new Locale.Builder()
                  .setRegion("JP")
                  .setLanguage("jp")
                  .build();
```

4. Using predefined static final constants

```
Locale locale4 = Locale.ITALIAN;
```

Resource Bundles

Resource bundles provide a solution to this problem of how to customize the application to locale-specific needs.


ResourceBundle (abstract class)
Resource bundles provide support for multiple locales; this class abstracts localization functionality
ListResourceBundle (abstract class)
PropertyResourceBundle
Manages resources for a locale as a list. For each locale, extend this class and override the getContents() method
Manages resources for a locale as property file. Use ResourceBundle.getBundle() method to automatically load relevant property file for a locale

PropertyResourceBundle

That name convention is:

package.Bundle_language_country_variant

For example:

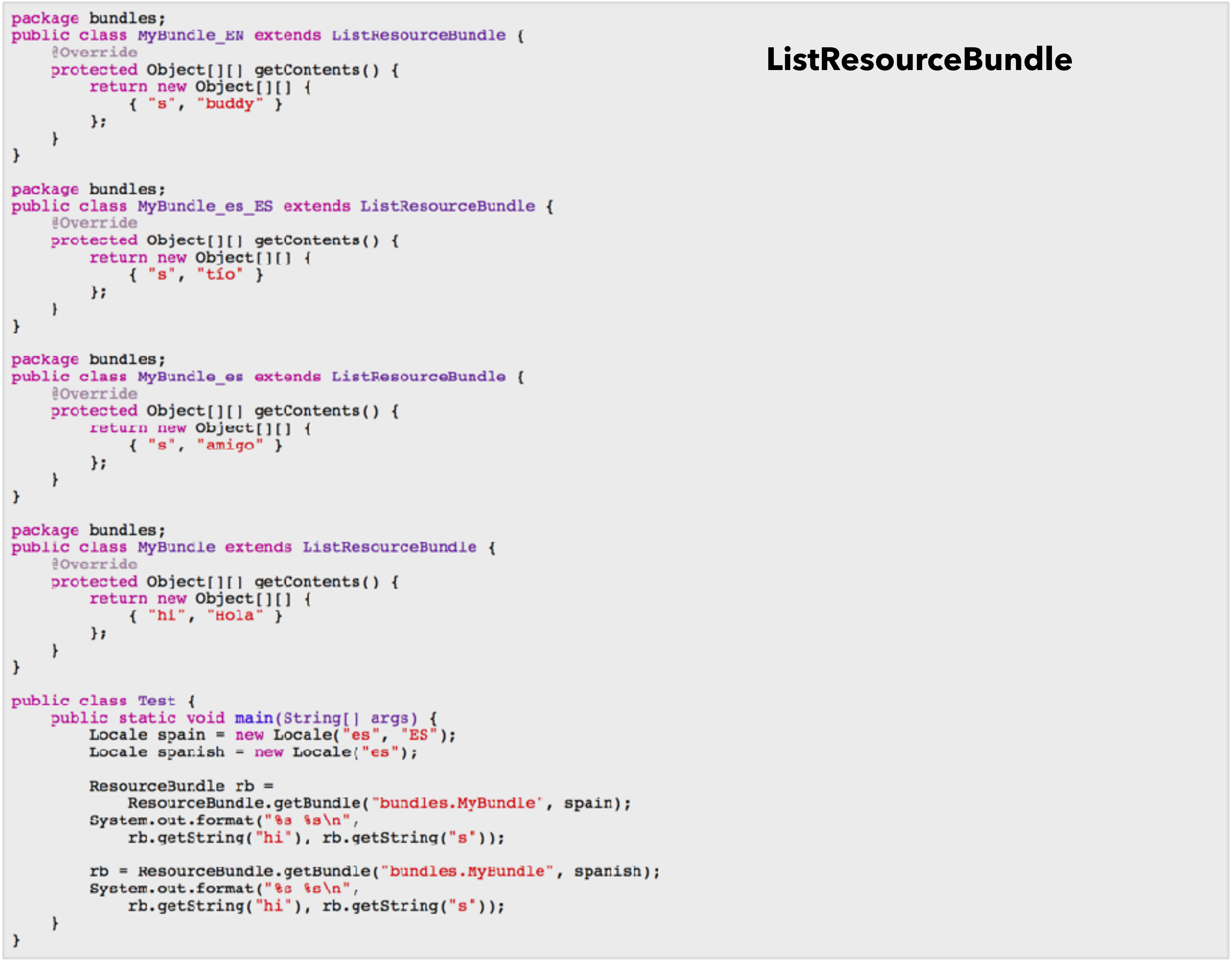
com.example.MyBundle_fr_FR

```
Zoo_en.properties
        hello=Hello
        open=The zoo is open.

Zoo_fr.properties
        hello=Bonjour
        open=Le zoo est ouvert
```

```
public static void main(String[] args) {
   Locale us = new Locale("en", "US");
   Locale france = new Locale("fr", "FR");

   printProperties(us);
   System.out.println();
   printProperties(france);
}

public static void printProperties(Locale locale) {
   ResourceBundle rb = ResourceBundle.getBundle("Zoo", locale);
   System.out.println(rb.getString("hello"));
   System.out.println(rb.getString("open"));
}
```

```
Hello
The zoo is open.

Bonjour
Le zoo est ouvert
```

ListResourceBundle



```java
package bundles;
public class MyBundle_EN extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "buddy" }
        };
    }
}

package bundles;
public class MyBundle_es_ES extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "tío" }
        };
    }
}

package bundles;
public class MyBundle_es extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "amigo" }
        };
    }
}

package bundles;
public class MyBundle extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "hi", "Hola" }
        };
    }
}

public class Test {
    public static void main(String[] args) {
        Locale spain = new Locale("es", "ES");
        Locale spanish = new Locale("es");

        ResourceBundle rb =
            ResourceBundle.getBundle("bundles.MyBundle", spain);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));

        rb = ResourceBundle.getBundle("bundles.MyBundle", spanish);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));
    }
}
```

Determining Which Resource Bundle to Use

```
ResourceBundle.getBundle("name");
ResourceBundle.getBundle("name", locale);
```

Step	Looks for File	Reason
1	Zoo_fr_FR.java	The requested locale
2	Zoo_fr_FR.properties	The requested locale
3	Zoo_fr.java	The language we requested with no country
4	Zoo_fr.properties	The language we requested with no country
5	Zoo_en_US.java	The default locale
6	Zoo_en_US.properties	The default locale
7	Zoo_en.java	The default language with no country
8	Zoo_en.properties	The default language with no country
9	Zoo.java	No locale at all—the default bundle
10	Zoo.properties	No locale at all—the default bundle
11	If still not found, throw MissingResourceException.	

Listing the parent resource bundles

```
Zoo.properties
name=Vancouver Zoo

Zoo_en.properties
hello=Hello
open=is open

Zoo_en_CA.properties
visitor=Canada visitor

Zoo_fr.properties
hello=Bonjour
open=est ouvert

Zoo_fr_CA.properties
visitor=Canada visiteur
```

```java
Locale locale = new Locale("en", "CA");
ResourceBundle rb = ResourceBundle.getBundle("Zoo", locale);
System.out.print(rb.getString("hello"));
System.out.print(". ");
System.out.print(rb.getString("name"));
System.out.print(" ");
System.out.print(rb.getString("open"));
System.out.print(" ");
System.out.print(rb.getString("visitor"));
```

The answer is `Hello. Vancouver Zoo is open Canada visitor.`

X