**Internationalization** is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term **internationalization** is abbreviated as **i18n**, because there are 18 letters between the first "i" and the last "n."

**Localization** is also abbreviated as I10N because there are total 10 characters between the first letter 'L' and last letter 'N'. Localization is the mechanism to create such an application that can be adapted to a specific language and region by adding locale-specific text and component.

There is the list of culturally dependent data:

* Messages
* Dates
* Times
* Numbers
* Currencies
* Measurements
* Phone Numbers
* Postal Addresses
* Labels on GUI components etc.

**Locale**

An object of Locale class represents a geographical or cultural region. This object can be used to get the locale specific information such as country name, language, variant etc.

Some examples for fields of Locale class:

1. public static final Locale ENGLISH
2. public static final Locale FRENCH
3. public static final Locale GERMAN
4. public static final Locale ITALIAN
5. public static final Locale JAPANESE
6. public static final Locale KOREAN
7. public static final Locale CHINESE
8. public static final Locale SIMPLIFIED\_CHINESE
9. public static final Locale TRADITIONAL\_CHINESE
10. public static final Locale FRANCE

Constructors of Locale class

There are three constructors of Locale class. They are as follows:

1. Locale(String language)
2. Locale(String language, String country)
3. Locale(String language, String country, String variant)

Commonly used methods of Locale class

There are given commonly used methods of Locale class.

1. **public static Locale getDefault()** it returns the instance of current Locale
2. **public static Locale[] getAvailableLocales()** it returns an array of available locales.
3. **public String getDisplayCountry()** it returns the country name of this locale object.
4. **public String getDisplayLanguage()** it returns the language name of this locale object.
5. **public String getDisplayVariant()** it returns the variant code for this locale object.
6. **public String getISO3Country()** it returns the three letter abbreviation for the current locale's country.
7. **public String getISO3Language()** it returns the three letter abbreviation for the current locale's language.

**Resource Bundle**

The **Resource Bundle class** is used to internationalize the messages. In other words, we can say that it provides a mechanism to globalize the messages.

Conventionally, the name of the properties file should be **filename\_languagecode\_country** code for example **MyMessage\_en\_US.properties**.

Commonly used methods of ResourceBundle class

There are many methods in the ResourceBundle class. Let's see the commonly used methods of the ResourceBundle class.

* **public static ResourceBundle getBundle(String basename)** returns the instance of the ResourceBundle class for the default locale.
* **public static ResourceBundle getBundle(String basename, Locale locale)** returns the instance of the ResourceBundle class for the specified locale.
* **public String getString(String key)** returns the value for the corresponding key from this resource bundle.

Example of Resource Bundle:

ResourceBundle bundle = ResourceBundle.getBundle("MessageBundle", Locale.US);

System.out.println("Message in "+Locale.US +":"+bundle.getString("greeting"));

//changing the default locale to Indonesia

Locale.setDefault(**new** Locale("in", "ID"));

bundle = ResourceBundle.getBundle("MessageBundle");

System.out.println("Message in "+Locale.getDefault()+":"+bundle.getString("greeting"));

**I18N with Date**

The format of the dates differs from one region to another that is why we internationalize the dates.

We can internationalize the date by using the **getDateInstance()** method of the **DateFormat** class. It receives the locale object as a parameter and returns the instance of the DateFormat class

Commonly used methods of DateFormat class for internationalizing date

There are many methods of the DateFormat class. Let's see the two methods of the DateFormat class for internationalizing the dates.

* **public static DateFormat getDateInstance(int style, Locale locale)** returns the instance of the DateFormat class for the specified style and locale. The style can be DEFAULT, SHORT, LONG etc.
* **public String format(Date date)** returns the formatted and localized date as a string.

Example of Internationalizing Date

In this example, we are displaying the date according to the different locale such as UK, US, FRANCE etc. For this purpose we have created the printDate() method that receives Locale object as an instance. The format() method of the DateFormat class receives the Date object and returns the formatted and localized date as a string.

**static** **void** printDate(Locale locale){

DateFormat formatter =DateFormat.getDateInstance(DateFormat.DEFAULT,locale);

Date currentDate=**new** Date();

String date=formatter.format(currentDate);

System.out.println(date+" "+locale);

}

**I18N with Time**

The display format of the time differs from one region to another, so we need to internationalize the time.

For internationalizing the time, the **DateFormat** class provides some useful methods.

The **getTimeInstance()** method of the DateFormat class returns the instance of the DateFormat class for the specified style and locale.

Example of Internationalizing Time

In this example, we are displaying the current time for the specified locale. The format() method of the DateFormat class receives date object and returns the formatted and localized time as a string. Notice that the object of Date class prints date and time both.

**static** **void** printTime(Locale locale){

DateFormat formatter=DateFormat.getTimeInstance(DateFormat.DEFAULT,locale);

Date currentDate=**new** Date();

String time=formatter.format(currentDate);

System.out.println(time+" in locale "+locale);

}

**I18N with Number**

The representation of the numbers differs from one locale to another. Internationalizing the numbers is good approach for the application that displays the information’s according to the locales.

The **NumberFormat** class is used to format the number according to the specific locale. To get the instance of the NumberFormat class, we need to call either **getInstance()** or **getNumberInstance()** methods.

Syntax of these methods is given below:

**public** **static** NumberFormat getNumberInstance(Locale locale)

**public** **static** NumberFormat getInstance(Locale locale)//same as above

Example of Internationalizing Number

In this example, we are internationalizing the number. The format method of the NumberFormat class formats the double value into the locale specific number.

**static** **void** printNumber(Locale locale){

**double** dbl=105000.3245;

 NumberFormat formatter=NumberFormat.getNumberInstance (locale);

  String number=formatter.format(dbl);

  System.out.println (number+" for the locale "+locale);

}

**I18N with Currency**

As we have internationalize the date, time and numbers, we can internationalize the currency also. The currency differs from one country to another so we need to internationalize the currency.

The **NumberFormat** class provides methods to format the currency according to the locale. The **getCurrencyInstance()** method of the NumberFormat class returns the instance of the NumberFormat class.

The syntax of the getCurrencyInstance() method is given below:

public static NumberFormat getCurrencyInstance(Locale locale)

Example of Internationalizing Currency

In this example, we are internationalizing the currency. The format method of the NumberFormat class formats the double value into the locale specific currency.

**static** **void** printCurrency(Locale locale){

**double** dbl=10500.3245;

 NumberFormat formatter=NumberFormat.getCurrencyInstance(locale);

  String currency=formatter.format(dbl);

  System.out.println(currency+" for the locale "+locale);

}