Internationalisation

Internationalisation

- L'internationalisation d'un programme consiste à ce qu'il soit exécutable en plusieurs langues sans modification du code : « internationalization » ***i18n***
  - avec l'ajout de données régionalisées, le même programme devient international.
  - Les éléments textuels ne sont pas codés en dur mais stockés en dehors du source et récupérés dynamiquement
  - il est possible de formater les dates et autres éléments suivant le contexte culturel de l'utilisateur
  - un programme internationalisé est très rapidement adaptable à une nouvelle langue

exemple

```java
public class NotI18N {

    static public void main(String[] args) {

        System.out.println("Hello.");
        System.out.println("How are you?");
        System.out.println("Goodbye.");
    }
}
```

- Problèmes pour internationaliser :
  - On ne connaît pas à l'avance la langue de l'utilisateur
  - Les codeurs ne sont pas des traducteurs
  - Les traducteurs ne sont pas des codeurs
- Solutions : fichiers textes pour les traductions

java.util.Locale

- Objet définissant une zone géographique, politique ou culturelle spécifique

Constructor Summary

`Locale(String language)` Construct a locale from a language code.
`Locale(String language, String country)` Construct a locale from language, country.
`Locale(String language, String country, String variant)` Construct a locale from language, country, variant.

- java.util.Locale
  - qques variables statiques :

Field Summary

static Locale	**CANADA** Useful constant for country.
static Locale	**CANADA_FRENCH** Useful constant for country.
static Locale	**CHINA** Useful constant for country.
static Locale	**CHINESE** Useful constant for language.
static Locale	**ENGLISH** Useful constant for language.
static Locale	**FRANCE** Useful constant for country.
static Locale	**FRENCH** Useful constant for language.
static Locale	**GERMAN** Useful constant for language.
static Locale	**GERMANY** Useful constant for country.
static Locale	**ITALIAN** Useful constant for language.
static Locale	**ITALY** Useful constant for country.
static Locale	**JAPAN** Useful constant for country.
static Locale	**JAPANESE** Useful constant for language.
static Locale	**KOREA** Useful constant for country.
static Locale	**KOREAN** Useful constant for language.
static Locale	**PRC** Useful constant for country.
static Locale	**SIMPLIFIED_CHINESE**

java.util.ResourceBundle

- Objet contenant une collection d'objets spécifiques à une région (le plus souvent des Strings)
- On le charge en passant en paramètre le nom de la ressource (un fichier properties) et un objet de type ***Locale***

```
Locale currentLocale = new Locale(" fr","FR");
ResourceBundle myResources =
    ResourceBundle.getBundle("MessagesBundle",
currentLocale);
```

MessagesBundle.properties

```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

MessagesBundle_fr_FR.properties

greetings = Bonjour.

farewell = Au revoir.

inquiry = Comment allez-vous?

MessagesBundle_de_DE.properties

```
greetings = Hallo.
farewell = Tschüß.
inquiry = Wie geht's?
```

MessagesBundle_en_US.properties

```
greetings = Hello.
farewell = Goodbye.
inquiry = How are you?
```

Après internationalisation

```
import java.util.Locale;
import java.util.ResourceBundle;

public class I18NSample {

    static public void main(String[] args) {

        String language,country;

        if (args.length != 2) {
            language = new String("en");
            country = new String("US");
        } else {
            language = new String(args[0]);
            country = new String(args[1]);
        }

        Locale currentLocale;
        ResourceBundle messages;

        currentLocale = new Locale(language, country);

        messages = ResourceBundle.getBundle("MessagesBundle",currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
```

Exécution

```
% java I18NSample fr FR
Bonjour.
Comment allez-vous?
Au revoir.

% java I18NSample en US
Hello.
How are you?
Goodbye.
```

Caractères spéciaux

- Les fichiers properties doivent être écrit avec Unicode pour les caractères spéciaux.

- 2 options :
  - édition via l'éditeur Eclipse des fichiers properties (fastidieux et non pérenne)
  - Le programme **native2ascii** (dans JDK/bin) :

```
native2ascii fichierDepart fichierUnicode
```

Remarques

- Reste un problème :
  - Comment faire pour ne pas passer la localisation comme un argument du programme ?
- Solution :
  - Utiliser les propriétés de la classe System

exemple

```
import java.util.Locale;
import java.util.ResourceBundle;

public class I18NSample {

    static public void main(String[] args) {

        String language,country;
        language = System.getProperty("user.language");
        country = System.getProperty("user.country");

        Locale currentLocale;
        ResourceBundle messages;

        currentLocale = new Locale(language, country);

        messages = ResourceBundle.getBundle("MessagesBundle",currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
```

ou mieux

Method Detail

getDefault

```
public static Locale getDefault()
```

Gets the current value of the default locale for this instance of the Java Virtual Machine.

The Java Virtual Machine sets the default locale during startup based on the host environment. It is used by many locale-sensitive methods if no locale is explicitly specified. It can be changed using the `setDefault` method.

Returns:
the default locale for this instance of the Java Virtual Machine

getAvailableLocales

```
public static Locale[] getAvailableLocales()
```

Returns an array of all installed locales. The array returned must contain at least a `Locale` instance equal to `Locale.US`.

Returns:
An array of installed locales.

getISOCountries

```
public static String[] getISOCountries()
```

Returns a list of all 2-letter country codes defined in ISO 3166. Can be used to create Locales.

---

getISOLanguages

exemple

```
import java.util.Locale;
import java.util.ResourceBundle;

public class I18NSample {

    static public void main(String[] args) {

        Locale currentLocale = Locale.getDefault();

        ResourceBundle messages;

        messages = ResourceBundle.getBundle("MessagesBundle",currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
```

% java I18NSample

Bonjour.

Comment allez-vous?

Au revoir.

Remarques diverses

- Créer un objet Locale est simple : il suffit d'utiliser les bons codes
- Mais, cela ne suffit pas : c'est juste un objet que l'on passe à d'autres qui font le « vrai » travail : les objets « *locale-sensitive* »
- Les objets « *locale-sensitive* » ne savent pas forcément traiter tous les codes qu'on leur passent
- Pour connaître quels types de Locale ces objets peuvent traiter, on utilise la méthode *getAvailableLocales* sur ces objets.

Exemple : sur DateFormat

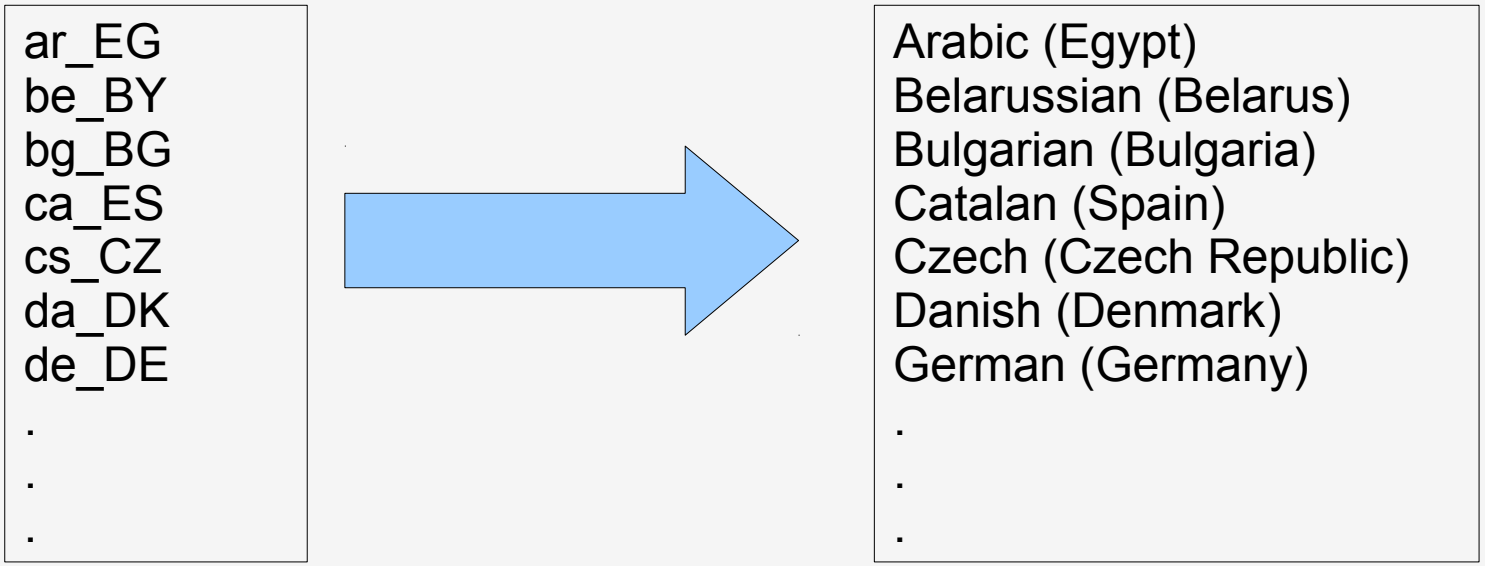
```
import java.util.*;
 import java.text.*;
 public class Available {
     static public void main(String[] args) {
  Locale[] list = DateFormat.getAvailableLocales();
  for (Locale aLocale : list) {
      System.out.println(aLocale.toString());
  }
     }    }
```

?

ar_EG
be_BY
bg_BG
ca_ES
cs_CZ
da_DK
de_DE
.
.
.

Présentation « end-users »

- Les codes ne sont pas forcément simples à comprendre, surtout pour les utilisateurs finaux.
- Pour avoir une présentation plus adaptée il faut utiliser la méthode suivante de l'objet Locale :
  - public final String getDisplayName()



Remarques diverses

- On n'est pas obligé d'utiliser un seul objet Locale dans l'intégralité du programme
- On peut définir un objet Locale par objet Locale-sensitive → Programmes multilingues.
- Problématiques client serveur :
  - I18n, côté serveur vs. côté client ?

Autres exemples

```
Date today = new Date();
DateFormat dateFormatter =
  DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);
String dateOut = dateFormatter.format(today);
System.out.println(dateOut + " " + currentLocale.toString());
```

DateFormat est Locale-sensitive

9 avr 98	fr_FR
9.4.1998	de_DE
09-Apr-98	en_US

Style	U.S. Locale	French Locale
DEFAULT	10-Apr-98	10 avr 98
SHORT	4/10/98	10/04/98
MEDIUM	10-Apr-98	10 avr 98
LONG	April 10, 1998	10 avril 1998
FULL	Friday, April 10, 1998	vendredi, 10 avril 1998

Autres exemples

```
DateFormat timeFormatter =
   DateFormat.getTimeInstance(DateFormat.DEFAULT,currentLocale)
   ;
```

Style	U.S. Locale	German Locale
DEFAULT	3:58:45 PM	15:58:45
SHORT	3:58 PM	15:58
MEDIUM	3:58:45 PM	15:58:45
LONG	3:58:45 PM PDT	15:58:45 GMT+02:00
FULL	3:58:45 oclock PM PDT	15.58 Uhr GMT+02:00

Autres exemples

```
DateFormat formatter = DateFormat.getDateTimeInstance
  (DateFormat.LONG, DateFormat.LONG, currentLocale);
```

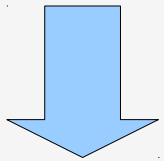
Style	U.S. Locale	French Locale
DEFAULT	25-Jun-98 1:32:19 PM	25 jun 98 22:32:20
SHORT	6/25/98 1:32 PM	25/06/98 22:32
MEDIUM	25-Jun-98 1:32:19 PM	25 jun 98 22:32:20
LONG	June 25, 1998 1:32:19 PM PDT	25 juin 1998 22:32:20 GMT+02:00
FULL	Thursday, June 25, 1998 1:32:19 o'clock PM PDT	jeudi, 25 juin 1998 22 h 32 GMT+02:00

Tester des caractères

```
char ch;
//...

// This code is WRONG!

if ((ch >= 'a' && ch <= 'z') ||
    (ch >= 'A' && ch <= 'Z'))
    // ch is a letter
// ...
if (ch >= '0' && ch <= '9')
  // ch is a digit
// ...
if ((ch == ' ') || (ch =='\n') || (ch == '\t'))
  // ch is a whitespace
```



```
// This code is OK!

if (Character.isLetter(ch))
// ...
if (Character.isDigit(ch))
// ...
if (Character.isSpaceChar(ch))

}
```