



# Java pour le développement de clients lourds

## Introduction à l'API « i18n »

Mickaël BARON - 2007 (Rév. Janvier 2009)  
<mailto:baron.mickael@gmail.com> ou <mailto:baron@ensma.fr>

# Licence

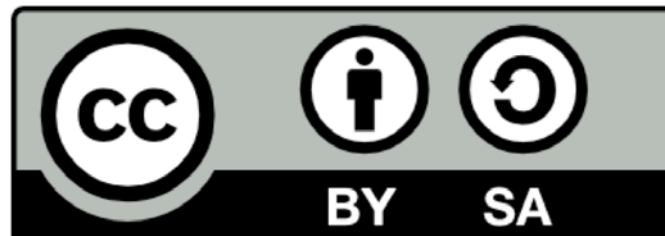
---

## Creative Commons

*Contrat Paternité*

*Partage des Conditions Initiales à l'Identique*

2.0 France



<http://creativecommons.org/licenses/by-sa/2.0/fr>

## Internationalisation : constats ...

---

- Une application de n'importe quel type doit pouvoir fournir une présentation des informations conformément à la localisation
- Selon la localisation les informations ne sont pas formatées de la même manière
- Les informations sont de plusieurs types
  - Chaînes de texte : « Bonjour », « Hello »
  - Heure : « 15:34 », « 3:34pm »
  - Date : « 12 août 2005 », « August 5, 2005 »
- La représentation des informations dépend aussi du contenu
  - Identification des informations : date, heure, nombre
  - Aspects grammaticaux : accords avec les éléments au pluriel

## Internationalisation : ... solution

---

- L'API i18n permet de gérer les aspects liés à la prise en compte de l'internationalisation
- Elle est basée sur un ensemble de classes et plus précisément
  - *java.util.Locale* : choix de la localisation
  - *java.util.ResourceBundle* : accéder aux données des fichiers localisation
- A cela s'ajoute des classes de manipulations de chaînes de caractères
  - *java.text.MessageFormat* : formatage des données
  - *java.text.ChoiceFormat* : utiliser pour gérer les différences

## Internationalisation : Locale

---

- Un objet de type *Locale* permet de donner les informations concernant la localisation de l'application
- Ces informations doivent être conformes aux réglementations du code de langage ISO
- Une localisation est constituée des éléments suivants
  - Langage : fr, en, sp, de, ...
  - Pays : FR, CA pour les pays francophones
  - Variant : WIN spécificités du système par exemple
- Des informations concernant cette réglementation du code de langage sont disponibles à cette adresse :
  - [www.loc.gov/standards/iso639-2/englangn.html](http://www.loc.gov/standards/iso639-2/englangn.html)
  - [www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html](http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html)

# Internationalisation : Locale

---

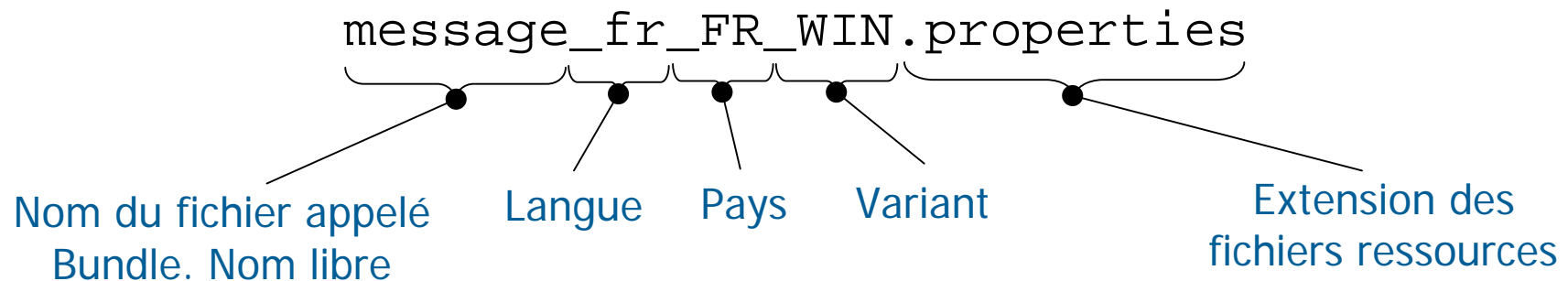
- Un objet de type *Locale* peut-être construit
  - *Locale(String language)* : en indiquant le code de langage
  - *Locale(String language, String country)* : en ajoutant le code du pays
  - *Locale(String lang, String coun, String variant)* : le système en plus

```
Locale my_french_locale = new Locale("fr", "FR");  
Locale my_english_locale = new Locale("en", "EN");
```

- Par défaut Java est à même d'identifier pour sa machine virtuelle la localisation de l'application (information système)
- Possibilité de connaître ou changer explicitement la localisation par défaut d'une application
  - *static Locale getDefault()* : retourne la locale de l'application
  - *static void setDefault(Locale)* : modifie la locale par défaut

# Internationalisation : ResourceBundle

- Un objet de type *ResourceBundle* permet de gérer un ensemble de fichier « propriétés » contenant les ressources localisées
- Les noms des fichiers « propriétés » doivent suivre une logique rigoureuse d'écriture dépendant du type de localisation



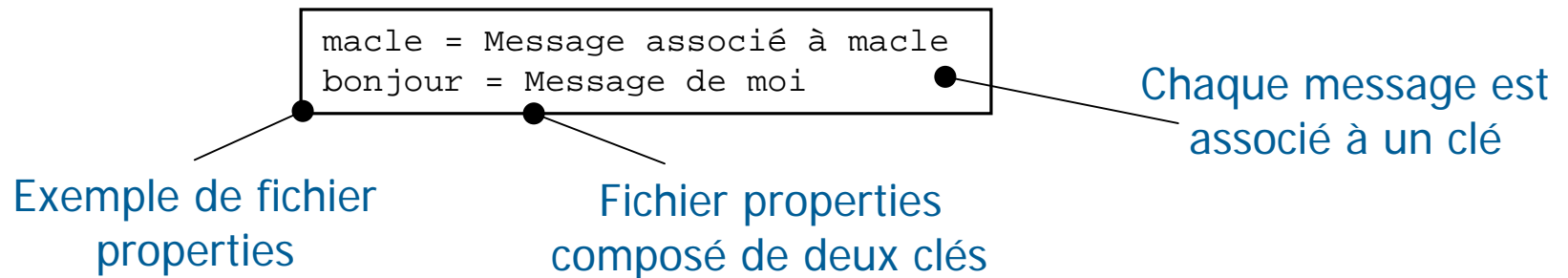
- Toutes les informations de la localisation sont séparées par un caractère obligatoire « \_ »



**Vous devez absolument respecter cette convention d'écriture**

# Internationalisation : ResourceBundle

- Un fichier properties contient un ensemble de message utilisable par le *ResourceBundle*
- Un message est associé à un clé spécifique de la forme :




- Suivant le type de locale, le *ResourceBundle* affiche le contenu exacte suivant la clé spécifiée

message\_fr.properties

```
macle = Message associé à macle
bonjour = Message de moi
```

message\_en.properties

```
macle = Associated message with macle
bonjour = Message from me
```



Il faut pour chaque clé proposer une correspondance de traduction

# Internationalisation : ResourceBundle

- Chaque objet *ResourceBundle* doit être associé à un fichier properties

Le choix du fichier properties (fonction de la localisation) est implicite



- Utilisation de la méthode *getBundle*

- *static ResourceBundle getBundle(String)* : objet *ResourceBundle* suivant un nom de fichier properties avec la locale par défaut
- *static ResourceBundle getBundle(String, Locale)* : *ResourceBundle* avec une locale explicite

**Le nom du bundle ne doit pas contenir l'extension, ni les informations liées à la locale**



- A partir de l'objet *ResourceBundle* il est possible de récupérer le contenu associé à la clé
  - *String getString(String cle)* : récupère le contenu du paramètre clé

# Internationalisation : ResourceBundle

## ➤ Exemple : modification de la locale en cours d'exécution

```
public class ChangeLocalisation {  
    public ChangeLocalisation() {  
        Locale my_french_locale = new Locale("fr");  
        Locale my_english_locale = new Locale("en");  
        ResourceBundle messages;  
        try {  
            messages = ResourceBundle.getBundle("message");  
            System.out.println(messages.getString("coucou"));  
            messages = ResourceBundle.getBundle("message",  
                my_english_locale);  
            System.out.println(messages.getString("coucou"));  
        } catch (MissingResourceException e) {  
            ... // Erreurs  
        }  
    }  
}
```

Définition de deux locales

Récupère le fichier bundle « message ». Par défaut la locale est « fr »

Affiche le contenu de la clé « coucou »

Récupère le fichier bundle « message ». En changeant la locale

coucou = hello

Fichier  
« message\_en.properties »

coucou = bonjour

Fichier  
« message\_fr.properties »

```
Problems Javadoc Declaration Console  
<terminated> ChangeLocalisation [Java Application] C:  
bonjour  
hello
```

# Internationalisation : MessageFormat

---

- Il peut être intéressant de paramétrer les messages contenus dans les fichiers properties. Exemple :
  - « Il est 13h45 en ce jour du 5 août 2005 »
  - « 13h45 », « 5 août 2005 » sont variables et formatables
- L'utilisation d'un objet *MessageFormat* est indépendant à la notion de *ResourceBundle*
- Un objet *MessageFormat* se construit par
  - *MessageFormat(String)* : message à formater avec la locale par défaut
  - *MessageFormat(String, locale)* : identique sauf qu'il est possible de préciser la locale « 13h45 » ou « 1:45pm »
- Les paramètres sont transmis à la chaîne à formater au moyen de la méthode *format* de *MessageFormat*
  - *String format(Object obj)* : *obj* contient les arguments à transmettre au message (stockées dans un tableau)

# Internationalisation : MessageFormat

---

- Dans le fichier properties l'utilisation de telles ou telles variables ce fait en utilisant les accolades

```
bonjour=Bonjour je m'appelle {0}●
```

Utilisation de  
l'argument  
indiqué 0

- Il est possible d'enrichir la description de *MessageFormat* pour formater directement les données dans le fichier properties
  - *{argind}* : utilise l'argument à l'indice *argind*
  - *{argind, formtype}* : idem avec un type de format spécifique
  - *{argind, formtype, formstyle}* : idem avec le style de format
- Les types de format sont
  - *number, date, time* et *choice*
- Les styles de format sont
  - *short, medium, long, full, integer, currency,*
  - *percent*

# Internationalisation : MessageFormat

## ➤ Exemple : formatage de messages passés en argument

```
ResourceBundle messages = ResourceBundle.getBundle("message", new Locale("fr", "FR"));

DateFormat formatter_date = new DateFormat(messages.getString("datemessage"),
    new Locale("en", "EN"));
DateFormat formatter_time = new DateFormat(messages.getString("heuremessage"),
    new Locale("fr", "FR"));

Object[] messageArguments = {
    new Date(),
    "Raoul"
};

String output = formatter_date.format(messageArguments);
System.out.println(output);
output = formatter_time.format(messageArguments);
System.out.println(output);
```

Date affichée à l'anglaise et l'heure à la française



L'utilisation de *MessageFormat* dans Java EE (JSTL par exemple) est totalement transparente

datemessage = Un simple message qui affiche la date courant {0,date,long}  
heuremessage = et aussi l'heure courante {0,time,medium}. Bonjour {1}

Fichier message\_fr.properties

```
Problems Javadoc Declaration Console
<terminated> MonTest [Java Application] C:\Program Files\Java\jre1.5.0_02\bin\javaw.exe (5 août 2005)
Un simple message qui affiche la date courant August 5, 2005
et aussi l'heure courante 14:31:23. Bonjour Raoul
```

## Internationalisation : ChoiceFormat

---

- Dans certains cas, les messages doivent prendre en compte les accords du singulier et du pluriel
  - « Il existe **une forme** » ●
  - « Il existe **deux formes** »
- Pour généraliser la classe *ChoiceFormat* permet selon un indice d'afficher un texte voulu
  - Le jour 1 est celui de **Lundi**
  - Le jour 2 est celui du **Mardi**
- Comme pour *MessageFormat* nous montrerons uniquement la forme dite « pattern » qui est celle exprimée dans les fichiers properties

Selon si l'adjectif numéral cardinal porte le pluriel ou pas  
« formes » s'accorde

# Internationalisation : ChoiceFormat

- *ChoiceFormat* exploite la forme  $\{argind, choice, formstyle\}$
- Le champ *formstyle* est défini par la sous forme

```
[Limit Index][Limit Behavior][Choice Text]|...
```

- [Limit Index] désigne l'indice du message associé (un entier)
- [Limit Behavior] désigne des signes de comparaison (# pour l'égal et < > pour les inférieurs et supérieurs)
- [Choice Text] le message à afficher selon le résultat de la comparaison
- Le caractère « | » permet de séparer les différents valeurs des limites

```
Message {0,choice,0#Bonjour|1#Bonsoir|1<#Adieu}
```

Il s'agit du premier paramètre

Il s'agit d'un patern de type *ChoiceFormat*

Plusieurs valeurs possibles pour le paramètre 0 de type entier

# Internationalisation : ChoiceFormat

## ➤ Exemple : message avec différentes valeurs selon des indices

```
MessageFormat formatter = new
    MessageFormat(messages.getString("monmessage"), new Locale("fr", "FR"));

Object[] messageArguments = {
    new Integer(2), "Tortue", new Integer(2)
};

output = formatter.format(messageArguments);
System.out.println(output);
```

Premier paramètre : un indice

Deuxième paramètre : une chaîne de caractères

Troisième paramètre : un indice

```
monmessage = Coucou {0,choice,0#Monsieur|1#Madame|2#Mademoiselle} {1},
vous {2,choice,0#n\`avez aucun message|1#avez un message|2#avez plusieurs
messages}
```

Pattern placé dans les fichiers properties

```
<terminated> MonTest [Java Application] /System/Library/Frameworks
Home/hin/java (18 août 2005 13:03:22)
Coucou Mademoiselle Tortue, vous avez plusieurs messages
```

# Internationalisation : intégration développement

---

- Création des fichiers « properties » et les placer dans un répertoire ressources
- Création d'une classe « constante » qui regroupe toutes les chaînes de caractères utilisées dans la partie présentation
- Suivant le type de localisation (celui par défaut ou donnée explicitement) toutes les chaînes de caractères sont initialisées
- A l'initialisation de tous les composants graphiques de l'application les valeurs textuelles sont extraites de la classe constante
- Inconvénients de cette solution
  - Impossibilité de changer la localisation puisque tous les objets graphiques sont construits avec la localisation
  - Solution : reprendre tous les composants graphique et modifier la valeur texte