

Soft Blue

Java Avançado

Internacionalização

Soft Blue

www.softblue.com.br

Todos os direitos de cia reservados. No  permitida a distribuio fsica ou eletrnica deste material sem a permisso expressa e por escrito do autor.

Tpicos Abordados

- O que  internacionalizao
- A classe *Locale*
- Usando objetos *ResourceBundle*
- Mensagens compostas
- Formatao de acordo com a localidade
 - Nmero
 - Moeda
 - Data
 - Hora

Internacionalizao

-  o processo de criar uma aplicao de forma que ela consiga se adaptar a vrias lnguas e regies de forma automtica
- O termo abreviado  **i18n**

Text Editor

Arquivo

Editor de Textos

Internacionalização

- O que pode ser internacionalizado?

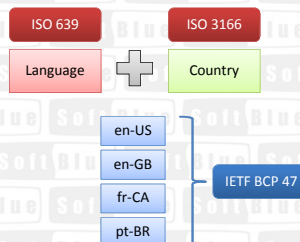
- Textos
 - "File" vs "Arquivo"
- Números
 - "1,540.32" vs "1.540,32"
- Moedas
 - "\$ 230.15" vs "R\$ 230,15"
- Datas
 - "12/22/2030" vs "22/12/2030"

Locale

- Um objeto *Locale* identifica uma combinação de língua e região (país)

Locale	
Language	Country
en	US
en	GB
fr	CA
pt	BR

Dados que representam um Locale



Criando Objetos Locale

- Construtores de *Locale*

```
Locale loc = new Locale("pt");
```

```
Locale loc = new Locale("pt", "BR");
```

- Método *forLanguageTag()* (a partir do Java 7)

```
Locale loc = Locale.forLanguageTag("pt-BR");
```

- Constantes pré-definidas

```
Locale loc = Locale.UK;
```

```
Locale loc = Locale.JAPAN;
```

Locale Padrão

- Toda aplicação Java tem um *Locale* padrão

- Ele é determinado pela JVM a partir de configurações definidas no sistema operacional

- É possível obter o *Locale* padrão via programação:

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

A Classe ResourceBundle

- Armazenam informações que podem variar de acordo com o *Locale* estabelecido

- As informações são baseadas em pares de chave e valor

- Dados podem ser armazenados de duas formas

- Arquivo de propriedades

- Chave do tipo *String*

- Valor do tipo *String*

- Classe

- Chave do tipo *String*

- Valor do tipo *Object*

Arquivos de Propriedades

Application.properties

```
# Itens do menu (inglês)
i1 = New
i2 = Open
i3 = Save
i4 = Exit
```

Application_pt_BR.properties

```
# Itens do menu (português)
i1 = Novo
i2 = Abrir
i3 = Salvar
i4 = Sair
```

Classe

```
public class Application extends ListResourceBundle {
    private static Object[][] contents = {
        { "i1", "New" },
        { "i2", "Open" },
        { "i3", "Save" },
        { "i4", "Exit" },
    };

    protected Object[][] getContents() {
        return contents;
    }
}
```

```
public class Application_pt_BR extends ListResourceBundle {
    private static Object[][] contents = {
        { "i1", "Novo" },
        { "i2", "Abrir" },
        { "i3", "Salvar" },
        { "i4", "Sair" },
    };

    protected Object[][] getContents() {
        return contents;
    }
}
```

Lendo os Dados

- Criar um *Locale*

```
Locale l = new Locale("pt", "BR");
```

- Obter uma instância de *ResourceBundle*

```
ResourceBundle bundle =
    ResourceBundle.getBundle("Application", l);
```

```
ResourceBundle bundle =
    ResourceBundle.getBundle("Application");
```

- Ler os dados a partir das chaves

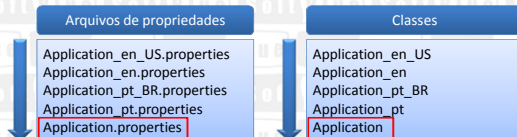
```
String s = bundle.getString("i1");
```

```
String s = (String) bundle.getObject("i1");
```

Ordem de Pesquisa

- Um *ResourceBundle* precisa definir de onde ele carregará as informações
- Exemplo
 - Bundle name = **Application**
 - Locale = **en-US**
 - Default Locale = **pt-BR**

Sempre defina um arquivo/classe sem sufixo



Mensagens Compostas

- Mensagens às vezes precisam ser compostas por partes fixas e variáveis

Português Fui ao shopping e comprei **2 camisas**. Custou **RS 200.00**.

Inglês I went to the mall and bought **2 shirts**. It cost **\$ 200.00**.

Application_pt_BR.properties

msg = Fui ao shopping e comprei {0,number} {1}. Custou {2,number,currency}.

i = camisas

Application_en_US.properties

msg = I went to the mall and bought {0,number} {1}. It cost {2,number,currency}.

i = shirts

Mensagens Compostas

- Criar o *Locale* e o *ResourceBundle*

```
Locale loc = new Locale("pt", "BR");
ResourceBundle bundle =
    ResourceBundle.getBundle("Application", loc);
```

- Criar o array de parâmetros

```
Object[] msgArgs = {
    new Integer(2),
    bundle.getString("i"),
    200.0
};
```

- Gerar a mensagem final

```
String msgTemplate = bundle.getString("msg");
MessageFormat formatter = new MessageFormat(msgTemplate);
formatter.setLocale(loc);
String msg = formatter.format(msgArgs);
```

Formatação de Números

- Diferentes localizações representam números e moedas de formas diferentes

```
NumberFormat nf = NumberFormat.getNumberInstance(  
    new Locale("pt", "BR"));  
nf.format(1540.36);
```

1.540,36

```
NumberFormat nf = NumberFormat.getNumberInstance(  
    new Locale("en", "US"));  
nf.format(1540.36);
```

1,540.36

Formatação de Moedas

- As moedas também variam de acordo com a localização

```
NumberFormat nf = NumberFormat.getCurrencyInstance(  
    new Locale("pt", "BR"));  
nf.format(1540.36);
```

R\$ 1.540,36

```
NumberFormat nf = NumberFormat.getCurrencyInstance(  
    new Locale("en", "US"));  
nf.format(1540.36);
```

\$1,540.36

Formatação de Datas

- Datas são representadas de formas diferentes em diferentes localizações

```
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG,  
    new Locale("pt", "BR"));  
df.format(new Date());
```

22 de Setembro de 2030

```
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG,  
    new Locale("en", "US"));  
df.format(new Date());
```

September 22, 2030

Formatação de Horas

- Horas também são representadas de formas diferentes em diferentes localizações

```
DateFormat df = DateFormat.getInstance(DateFormat.MEDIUM,  
    new Locale("pt", "BR"));  
df.format(new Date());
```

15:34:45

```
DateFormat df = DateFormat.getInstance(DateFormat.Medium,  
    new Locale("en", "US"));  
df.format(new Date());
```

3:34:45 PM

Colocando em Prática...



Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

[Clique aqui para acessar as vídeo-aulas práticas](#)
