03

Resumo

Transcrição

Vamos rever os últimos conteúdos? Primeiro trabalhamos com datas, e descobrimos que o Java gera datas no padrão internacional com o `LocalDate`. Criamos alguns formatadores com o `ofPattern()`, que recebe uma string para estabelecer o padrão a ser seguido. Escolhemos o formato `"dd/MM/yyyy"`. Foi criado o `hoje` como `LocalDate`, e nele foi aplicado esse padrão, com o `DateTimerFormatter`.

Depois, criamos um `LocalDateTime`, o `agora`. Ele inclui também o horário local com minutos, e definimos o formato `"dd/MM/yyyy hh:mm"` com o `DateTimerFormatter`.

Trabalhamos também com o outro formatador, o `FormatStyle`, associado ao `withLocale`, que recebe um `Locale()`. Criamos o `Locale("pt", "br")`, pois não havia uma constante, dentre as muitas que vimos ao clicar na classe, que nos servisse. O `Locale()` é importante para que o padrão escolhido se mantenha mesmo em uma máquina configurada em outra língua.

Aprendemos também a adicionar e a remover horas, usando respectivamente o `plusHours` e o `minusHours`. Ainda é possível usar o `plusDays` e `plusYears`, caso seja necessário.

Todos esses aprendizados resultaram nesse trecho de código:

```
package br.com.alura;

public class Data {

  public static void main(String [] args) {
    LocalDate hoje = LocalDate.now();
    System.ou.println(hoje);
    DateTimerFormatter formatador = DateTimerFormatter.ofPattern("dd/MM/yyyy");
    System.out.println(hoje.format(formatador));
    LocalDateTime agora = LocalDateTime.now();
    System.out.println(agora);
    DateTimerFormatter formatadorHora = DateTimerFormatter.ofPattern("dd/MM/yyyyy hh:mm");
    System.out.println(agora.format(formatadorHora));
    DateTimerFormatter formatadorCurto = DateTimerFormatter.ofLocalizedDateTime(formatStyle.MED
    System.out.println(agora.format(formatadorCurto));
    System.out.println(agora.minusHours(5));
    System.out.println(agora.plusHours(10));
  }
}
```

Que gera os seguintes valores:

```
2016-11-07
07/11/2016
2016-11-07T11:08:13.273
07/11/2016 11:08
```

```
2016-11-07T06:08:13:273
2016-11-07T21:08:13:273
```

Outra coisa que aprendemos foi trasformar algarismos em números escritos por extenso. Usamos a classe `inWords` do Stella. Criamos um `valor` para poder escrevê-lo por extenso. Escolhemos usar o valor como `BigDecimal` em vez de `double` por se tratar de dinheiro e o arredondamento em `double` não é muito bom. Com a classe `NumericToWordsConverter`, obtivemos um conversor. Dentro dele, usamos o método `toWords`, que pega o valor (transformado em `double` a esse nível apenas) e o transforma em palavras.

```
package br.com.alura;

public class NumeroPorExtenso {

  public static void main(String[] args){
    NumericToWordsConverter coversor = NumericToWordsConverter(new FormatoDeReal());
    BigDecimal valor = new BigDecimal("900.59");
    String valorPorExtenso = conversor.toWords(valor.doubleValue());
    System.out.println(valorPorExtenso);

  }
}
```

O console nos exibe o seguinte:

```
novecentos reais e cinquenta e nove centavos
```

Como usamos também o `FormatoDeReal`, o programa reconhecerá até os centavos e adequará singular e plural aos valores inseridos, o que é bem útil em contratos. Como exemplo de aplicação, vimos o próprio site da Alura. Até a próxima!