01

Formatação data com o padrão brasileiro

Transcrição

Outra peculiaridade brasileira é a formatação da data, que não segue o padrão mais comum internacionalmente. Nós usamos o padrão `DD/MM/AAAA`, que deve estar presente quando desenvolvemos aplicação no Brasil.

Para formatar datas, criaremos uma classe nova. Clicaremos no pacote `br.com.alura` com o botão direito, e em seguida em `New > Class`. Ela se chamará `Data`. A primeira coisa que faremos nessa classe é criar o método `main`.

```
package br.com.alura;

public class Data {

  public static void main(String [] args) {

  }
}
```

No Java 8 ficou um pouco mais fácil lidar com datas. Essa parte foi totalmente reformulada com base em um framework muito usado pela comunidade, o `Java time`. Pegaram dele as peculiaridades de data e colocaram na nova API do Java. A classe que usamos para criar uma nova data é o `LocalDate`, e o método é `now()`.

```
package br.com.alura;

public class Data {

  public static void main(String [] args) {
    LocalDate.now();
  }
}
```

Ele gerará a data de agora. Criaremos uma variável para essa data, cujo nome será `hoje`. E pediremos para printar, para que vejamos seu formato.

```
package br.com.alura;

public class Data {

  public static void main(String [] args) {
    LocalDate hoje = LocalDate.now();
    System.ou.println(hoje);
  }
}
```

E vamos rodar, com o botão direito e a seguir `Run as > Java Application`. O console nos retorna o seguinte:

```
2016-11-07
```

A data veio correta, mas está em formato internacional `aa-MM-dd`. Para colocar em formato brasileiro no Java 7, usávamos `SimpleDateFormat` e faríamos a formatação. Com o Java 8 ficou mais fácil, usando a classe `DateTimeFormat`, que tem o método `ofPattern()`, que recebe uma string e coloca no padrão que desejamos, a ser colocado nos parênteses. Será `dd/MM/yyyy`. Note que o mês é representado pelo `M` maiúsculo. O objeto dessa instância será `formatador`.

```
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");
  }
}
```

Agora pediremos o `sysout` da data nesse novo formato, passando o `formatador`.

```
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));
  }
}
```

O console nos devolve o seguinte:

```
2016-11-07
07/11/2016
```

Conseguimos passar com sucesso o `formatador` pelo método `format`.

E se quisermos trabalhar com hora? Temos a classe `LocalDateTime`, que também possui o método `now()`. A variável será `agora`. Já pediremos o `sysout`.

```
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);
  }
}
```

Teremos o seguinte:

```
2016-11-07
07/11/2016
2016-11-07T10:47:57.251
```

Ele nos mostra a data, seguida de um `T` que a separa da data, que é precisada até os nanossegundos. O formato que mais nos interessa é o `dd/MM/yyyy hh:mm` , agora com o `m` minúsculo por se tratarem de minutos. O formatador agora se chamará `formatadorHora` , para diferenciá-lo do anterior. Já vamos pedir para printar.

```
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));
  }
}
```

Ao rodar, teremos:

```
2016-11-07
07/11/2016
2016-11-07T10:47:57.251
07/11/2016 10:49
```

Agora está em um formato que nos agrada, que é o padrão aqui no Brasil. Com a classe `DateTimerFormatter` conseguimos formatar com a data e a hora com sucesso.

Outro método do `DateTimerFormatter` é o `ofLocalizedDateTime` , que recebe um formato em `style` , que é um `enum` do Java 8. Ele formata a data sem que precisemos passar um `ofPattern` Vamos ver como é?

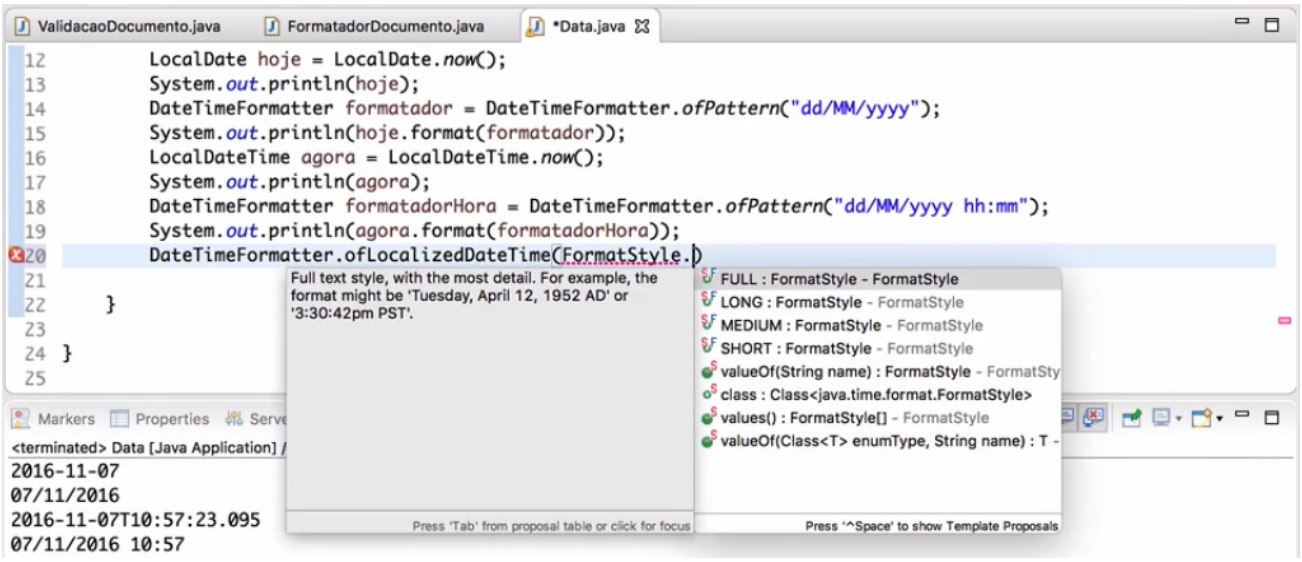
```
package br.com.alura;

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.ofLocalizedDateTime(formatStyle.SHORT);
  }
}
```

O argumento desse `enum` é o `formatStyle`, que pode ser `SHORT` ou `MEDIUM`, se queremos trabalhar com hora. As opções aparecem da seguinte maneira:



Vamos criar uma instância para ele, que será o `formatadorCurto`.

```
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.SHOI
  }
}
```

Faremos um `sysout` com esse formatador, usando o `agora` que criamos há pouco.

```
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.SHOI
    System.out.println(agora.format(formatadorCurto));
  }
}
```

Ao pedir para o programa rodar, temos:

```
2016-11-07
07/11/2016
2016-11-07T11:02:37.340
07/11/2016 11:02
07/11/16 11:02
```

A data está formatada sem precisarmos especificar um padrão, pois ele já está pronto, o `SHORT`. Vamos ver como é o `MEDIUM`.

```
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.ofLocalizedDate(formatStyle.SHORT);
    System.out.println(agora.format(formatadorCurto));
    DateTimerFormatter formatadorCurto = DateTimerFormatter.ofLocalizedDateTime(formatStyle.MEDI
    System.out.println(agora.format(formatadorCurto));
  }
}
```

O console nos retorna o seguinte:

```
2016-11-07
07/11/2016
2016-11-07T11:02:37.340
07/11/2016 11:02
07/11/16 11:02
07/11/2016 11:02:59
```

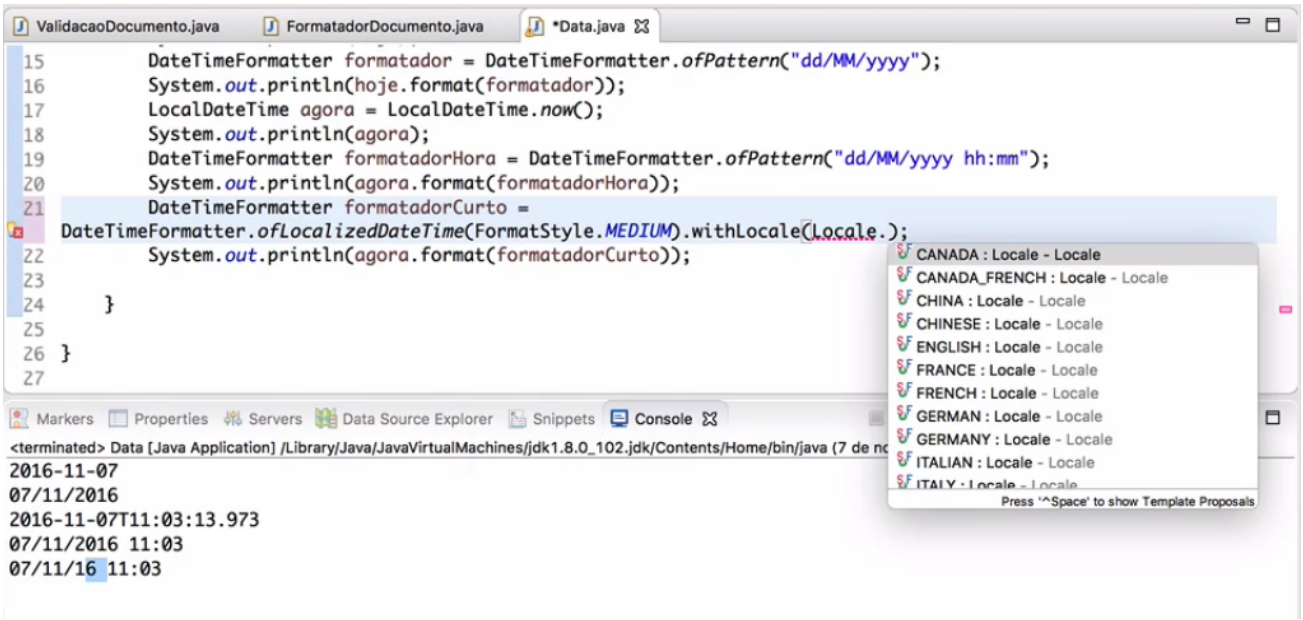
Observe as diferenças: no `MEDIUM`, o ano aparece com 4 dígitos e o horário inclui os segundos. Eu acho esse formato mais interesse, e é esse que manteremos no código. Outra opção que o `ofLocalizedDateTime` nos dá é o uso de um `withLocale`, que é um local associado à data. Também é um `Enum`.

```
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));
  }
}
```

O argumento do `withLocale` é um `Locale`. O próprio Eclipse nos dá algumas opções de lugar, quando adicionamos o `..`



Escolheremos `CHINA`.

```
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));
  }
}
```

Usando o `sysout` , o console nos mostra:

```
2016-11-07
07/11/2016
2016-11-07T11:04:39.054
07/11/2016 11:04
2016-11-7 11:04:39
```

O ano veio primeiro e as barras foram substuídas por traços. Não é o padrão que mais nos agrada. Mas, se formos procurar dentre as opções, não encontramos nada do Brasil. Para usar a formatação brasileira, teremos que criar. Faremos isso criando um `new Locale` inserindo dois argumentos: a língua `" pt"` e qual sua especificação `"br"`.

```
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));
  }
}
```

Ao rodar, o console nos mostra o seguinte:

```
2016-11-07
07/11/2016
2016-11-07T11:05:22.573
07/11/2016 11:05
07/11/2016 11:05:22
```

Voltou ao padrão anterior. Apesar disso, é sempre bom especificar a língua. Podemos estar rodando o programa com o sistema operacional em português, mas se ele for usado em um lugar com o sistema operacional em inglês, o `Locale` será puxado de lá. Isso pode gerar um problema na hora de colocar o seu código em produção. Assim, recomenda-se setar o `Locale` para o Brasil com o `withLocale` .

Podemos entrar na classe apertando o `Ctrl` e clicando sobre ela. Veremos que ela tem várias constantes:

```
...
public Locale(String language, String country) {
  this(language, country, "")
}

...

/** Useful constant for language.
 */
static public final Locale ENGLISH = createConstant("en", "");

/** Useful constant for language.
 */
static public final Locale FRENCH = createConstant("fr", "");

/** Useful constant for language.
 */
static public final Locale GERMAN = createConstant("de", "");
...
```

E se precisarmos adicionar ou remover horas do horário atual? Podemos fazer isso facilmente com essa API. Usaremos novamente a instância `agora` . Para remover 5 horas, usaremos o `minusHours()` e já mandaremos printar.

```
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));
```

```
    }
  }
```

E o console nos mostrará:

```
2016-11-07
07/11/2016
2016-11-07T11:07:41.628
07/11/2016 11:07
2016-11-07T06:07:41:628
```

Como `11 - 5 = 6` , vemos a conta foi feita corretamente. E se quisermos acrescentar? Basta usar o `plusHours()` . Acrescentaremos 10 horas.

```
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));
  }
}
```

Mandando rodar, temos:

```
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
```

Novamente, o resultado esta correto. Nessa API permite que você adicione anos, dias, minutos... Ela é bem fácil de lidar com essas somas ou subtrações. É uma API bem completa e tem vários métodos que podemos usar no cotidiano das aplicações brasileiras. Até a próxima!