01

A nova API de datas

Transcrição

Já vimos bastante das principais novidades da linguagem. Vamos conhecer agora mais uma grande introdução do java 8, a nova API de datas. Essa é uma API bem grande, que fazia bastante falta e foi por muito tempo esperada pelos desenvolvedores.

Se você já trabalha com Java sabe as dificuldades em trabalhar com as classes `Date` e `Calendar`. Se você ainda não conhece essas classes, não se preocupe, você já pode começar a aprender e utilizar os modelos novos.

Pra começar vamos criar uma classe simples com um método `main`, onde vamos fazer os nossos testes com a nova APi.

```
public class Datas {

    public static void main(String[] args) {

    }
}
```

Vamos começar criando uma data, a data de hoje. Para representar uma data em java agora eu posso utilizar a classe `LocalDate`, presente no pacote `java.time`. Repare como é facil ter a data atual utilizando o método `now()`:

```
public class Datas {

    public static void main(String[] args) {

        LocalDate hoje = LocalDate.now();
        System.out.println(hoje);

    }
}
```

Assim como este existem diversos outros métodos estáticos nas novas classes da API de datas. Vamos conhecer vários no decorrer do capítulo.

Repare que o valor impresso neste caso será `2014-05-28`, logo veremos como formatar essa saída de outras formas.

Conhecendo mais da API

Vamos agora criar uma nova data para representar as Olimpíadas do Rio, por exemplo. Para fazer isso podemos utilizar o método `of` passando o dia, mês e ano:

```
LocalDate olimpiadasRio = LocalDate.of(2016, Month.JUNE, 5);
```

Repare que existe uma enumeração pra representar o mês, mas se você preferir pode utilizar sim um numero inteiro.

Vamos agora calcular a diferença de anos entre essas duas datas. Uma forma de fazer isso na mão seria subtraindo o método `getYear` das datas, algo como:

```
int anos = olimpiadasRio.getYear() - hoje.getYear();
System.out.println(anos);
```

Trabalhando com Period

Ao executar esse código temos o resultado esperado, que neste caso é 2 anos. Mas e se quisessemos descobrir a diferença de dias e meses também? Daria pra fazer da mesma forma, mas sempre que você tiver que fazer um trabalho dessa na mão você pode ter certeza que já existe algo pronto pra te ajudar de alguma forma. Nesse caso podemos utilizar a classe `Period` .

Para saber a diferença entre duas datas podemos utilizar seu método `between` , da seguinte forma:

```
Period periodo = Period.between(hoje, olimpiadasRio);
System.out.println(periodo);
```

Repare que a saída desse nosso `println` vai ser um pouco estranha, um exemplo seria: `P2Y8D` .

Isso significa um período de 2 anos e 8 dias. Mas claro, poderíamos imprimir apenas as propriedades, da segunite forma:

```
Period periodo = Period.between(hoje, olimpiadasRio);
System.out.println(periodo.getYears());
System.out.println(periodo.getMonths());
System.out.println(periodo.getDays());
```

Assim como esses existem diversos getters pra passar informações importantes a respeito desse período.

Incrementando e decrementando suas datas

Outra coisa bem comum em nosso dia a dia é quando queremos saber o dia anterior ou posterior a uma data. Por exemplo como saber qual a data de amanhã? Há diversos métodos pra nos ajudar com isso, vamos encontrar na API diversos métodos `minus` ou `plus` pras diferentes unidades de tempo, como por exemplo:

```
System.out.println(hoje.minusYears(1));
System.out.println(hoje.minusMonths(4));
System.out.println(hoje.minusDays(2));

System.out.println(hoje.plusYears(1));
System.out.println(hoje.plusMonths(4));
System.out.println(hoje.plusDays(2));
```

Ou seja, pra saber a data de amanhã bastaria fazer `hoje.plusDays(1)` .

Uma API imutável

Sabendo disso podemos escrever o seguinte código para incrementar 4 anos na data atual, para saber quando será a próxima Olimpíada, por exemplo.

```
olimpiadasRio.plusYears(4);
System.out.println(olimpiadasRio);
```

Mas repare que a saída desse código ainda será a data atual. Porque isso ocorreu? Da mesma forma que as novas API's, como o Stream, os métodos da API de datas sempre vão retornar uma nova instancia da sua data. Portanto precisamos fazer algo como:

```
LocalDate proximasOlimpiadas = olimpiadasRio.plusYears(4);
System.out.println(proximasOlimpiadas);
```

Ou seja, toda a API de datas é imutável. Ela nunca vai alterar a data original.

Ao executar esse código, um exemplo da saída seria `2020-06-25` .

Mas note que não é esse o formato que estamos acostumados a trabalhar, podemos então trabalhar com os diversos formatadores de datas existentes.

Formatando suas datas

Para formatar nossas datas podemos utilizar o `DateTimeFormatter` . Existem diversos já prontos, mas há ainda a alternativa de você criar o seu próprio formatador no padrão já conhecido de `dd/MM/yyyy` .

Para fazer isso basta você utilizar o método `ofPattern` :

```
DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy");
```

Agora podemos a partir da nossa data, neste caso `proximasOlimpiadas` , chamar o método `format` passando esse formatador:

```
String valorFormatado = proximasOlimpiadas.format(formatador);
System.out.println(valorFormatado);
```

Agora sim, ao executar temos o resultado `05/06/2020`

Trabalhando com medida de tempo

Por enquanto só estamos trabalhando com datas, fazendo formatações e manipulando seu resultado. Mas é muito comum eu também precisar trabalhar com horas, minutos e segundos. Ou seja, trabalhar com uma medida de data com tempo.

Para isso podemos utilizar a classe `LocalDateTime` , de forma bem similar podemos fazer:

```
LocalDateTime agora = LocalDateTime.now();
```

Podemos criar um novo formatador para mostrar as horas, minutos e segundos para conseguirmos ver o resultado já formatado:

```
DateTimeFormatter formatadorComHoras = DateTimeFormatter.ofPattern("dd/MM/yyyy hh:mm:ss");
LocalDateTime agora = LocalDateTime.now();
System.out.println(agora.format(formatadorComHoras));
```

Pronto! Agora o resultado será algo como `05/06/2014 12:24:10` .

Lidando com modelos mais específicos

É muito comum ignorarmos valores quando precisamos apenas de algumas medidas de tempo, como por exemplo ano e mês. Nessa caso no lugar de criarmos um `LocalDate` ou algo assim e ignorar o seu valor de dia, podemos trabalhar com os modelos mais específicos da nova API.

Neste exemplo podemos usar o `YearMonth` , da seguinte forma:

```
YearMonth anoEMes = YearMonth.of(2015, Month.JANUARY);
```

Ou seja, existem diversas novas classes para expressar bem nossas intenções.

Outro exemplo, para trabalharmos apenas com tempo podemos utilizar o `LocalTime` . Representar o horario do nosso intervalo de almoço, por exemplo, poderia ser feito com:

```
LocalTime intervalo = LocalTime.of(12, 30);
System.out.println(intervalo);
```

Nesse caso a saída seria exatamente `12:30` . Não quer trabalhar com esse formato? Tudo bem, você pode criar um formatador como nós já fizemos, contanto que ele só tenha as medidas de tempo que existem, que neste caso são hora e minuto. Caso contrário uma exception será lançada.