CES/JF

Centro de Ensino Superior
de Juiz de Fora

Data e Hora no Java

Prof. Tassio Sirqueira
tassio@tassio.eti.br

Data e hora para máquinas

- A classe *Instant* é utilizada para o “agora”, com precisão de nanossegundos.
  - Instant agora = Instant.now();
  - System.out.println(agora); //2019-05-06T11:12:50.036Z (formato ISO-8601)
- Podemos usar um *Instant*, por exemplo, para medir o tempo de execução de um algoritmo.
  - Instant inicio = Instant.now();
  - Algoritmo();
  - Instant fim = Instant.now();
  - Duration duracao = Duration.between(inicio, fim);
  - long duracaoEmMilissegundos = duracao.toMillis();
- A classe Duration serve para medir uma quantidade de tempo em termos de nanossegundos. Essa quantidade de tempo pode ser alterada em diversas unidades, chamando métodos como toNanos, toMillis, getSeconds, etc.

Datas para humanos

- A classe *LocalDate* que representa uma data, ou seja, um período de 24 horas com dia, mês e ano definidos.
  - LocalDate hoje = LocalDate.now();
  - System.out.println(hoje); //2014-04-08 (formato ISO-8601)
- Um *LocalDate* serve para representarmos datas em que não nos importamos com as horas ou minutos, mas o dia todo.
- Para calcularmos a duração entre dois *LocalDate*, devemos utilizar um *Period*, que já trata anos bissextos e outros detalhes.
  - LocalDate homemNoEspaco = LocalDate.of(1961, Month.APRIL, 12);
  - LocalDate homemNaLua = LocalDate.of(1969, Month.MAY, 25);
  - Period periodo = Period.between(homemNoEspaco, homemNaLua);
  - System.out.printf("%s anos, %s mês e %s dias", periodo.getYears() , periodo.getMonths(), periodo.getDays()); //8 anos, 1 mês e 13 dias

Datas para humanos

- Já a classe *LocalTime* serve para representar apenas um horário, sem data específica. Podemos, por exemplo, usá-la para representar o horário de entrada no trabalho.
  - LocalTime horarioDeEntrada = LocalTime.of(9, 0);
  - System.out.println(horarioDeEntrada); //09:00
- A classe *LocalDateTime* serve para representar uma data e hora específicas.
  - LocalDateTime agora = LocalDateTime.now();
  - LocalDateTime aberturaDaCopa = LocalDateTime.of(2014, Month.JUNE, 12, 17, 0);
  - System.out.println(aberturaDaCopa); //2014-06-12T17:00 (formato ISO-8601)

Datas com fuso horário

- Para representarmos uma data e hora em um fuso horário específico, devemos utilizar a classe ZonedDateTime.
  - ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/Sao_Paulo");
  - ZonedDateTime agoraEmSaoPaulo = ZonedDateTime.now(fusoHorarioDeSaoPaulo);
  - System.out.println(agoraEmSaoPaulo); //2014-04-08T10:02:57.838-03:00[America/Sao_Paulo]

Datas com fuso horário

- Com um ZonedDateTime, podemos representar, por exemplo, a data de um voo.
  - ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/Sao_Paulo");
  - ZoneId fusoHorarioDeNovaYork = ZoneId.of("America/New_York");
  - LocalDateTime saidaDeSaoPauloSemFusoHorario = LocalDateTime.of(2014, Month.APRIL, 4, 22, 30);
  - LocalDateTime chegadaEmNovaYorkSemFusoHorario = LocalDateTime.of(2014, Month.APRIL, 5, 7, 10);
  - ZonedDateTime saidaDeSaoPauloComFusoHorario = ZonedDateTime.of(saidaDeSaoPauloSemFusoHorario, fusoHorarioDeSaoPaulo);
  - System.out.println(saidaDeSaoPauloComFusoHorario); //2014-04-04T22:30-03:00[America/Sao_Paulo]
  - ZonedDateTime chegadaEmNovaYorkComFusoHorario = ZonedDateTime.of(chegadaEmNovaYorkSemFusoHorario, fusoHorarioDeNovaYork);
  - System.out.println(chegadaEmNovaYorkComFusoHorario); //2014-04-05T07:10-04:00[America/New_York]
  - Duration duracaoDoVoo = Duration.between(saidaDeSaoPauloComFusoHorario, chegadaEmNovaYorkComFusoHorario);
  - System.out.println(duracaoDoVoo); //PT9H40M

Datas com fuso horário

- Outro cuidado importante que devemos ter é em relação ao horário de verão. No fim do horário de verão, por exemplo, a mesma hora existe duas vezes!
  - ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/Sao_Paulo");
-
  - LocalDateTime fimDoHorarioDeVerao2013SemFusoHorario = LocalDateTime.of(2014, Month.FEBRUARY, 15, 23, 00);
-
  - ZonedDateTime fimDoHorarioVerao2013ComFusoHorario = fimDoHorarioDeVerao2013SemFusoHorario.atZone(fusoHorarioDeSaoPaulo);
  - System.out.println(fimDoHorarioVerao2013ComFusoHorario); //2014-02-15T23:00-02:00[America/Sao_Paulo]
-
  - ZonedDateTime maisUmaHora = fimDoHorarioVerao2013ComFusoHorario.plusHours(1);
  - System.out.println(maisUmaHora); //2014-02-15T23:00-03:00[America/Sao_Paulo]

Formatando datas

- O *toString* padrão das classes da API utiliza o formato ISO-8601. Se quisermos definir o formato de apresentação da data, devemos utilizar o método *format*, passando um *DateTimeFormatter*.
  - LocalDate hoje = LocalDate.now();
  - DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy");
  - hoje.format(formatador); //08/04/2014

Formatando datas

- O *enum FormatStyle* possui alguns formatos pré-definidos, que podem ser combinados com um *Locale*.
  - LocalDateTime agora = LocalDateTime.now();
  - DateTimeFormatter formatador = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT).withLocale(new Locale("pt", "br"));
  - agora.format(formatador); //08/04/14 10:02

Referências

- Costa, R. **Data e Hora no Java 8 e no Java 9**. https://medium.com/@racc.costa/data-e-hora-no-java-8-e-no-java-9-5f1e3fd8d560. Acessado em 06/05/19
- Aquiles, A. e Ferreira, R. **Conheça a nova API de datas do Java 8**. https://blog.caelum.com.br/conheca-a-nova-api-de-datas-do-java-8/. Acessado em 06/05/19
- Normandes. **Introdução à nova API de Datas do Java 8.** https://blog.algaworks.com/introducao-a-nova-api-de-datas-do-java-8/. Acessado em 06/05/19