**Capítulo VINTE E DOIS**  
**Fusos Horários e Horário de Verão**

**Objetivos do Exame**  
Trabalhar com datas e horários em diferentes fusos horários e gerenciar mudanças resultantes do horário de verão, incluindo a formatação de valores de data e hora.

**Classes centrais de fuso horário**

Antes do Java 8, se quiséssemos trabalhar com informações de fuso horário, tínhamos que usar a classe java.util.TimeZone. Agora, com a nova API de Data/Hora, existem opções novas e melhores.

Elas são:

* **ZoneID**  
  Representa o ID do fuso horário. Por exemplo, Asia/Tokyo.
* **ZoneOffset**  
  Representa um deslocamento de fuso horário. É uma subclasse de ZoneId. Por exemplo, -06:00.
* **ZonedDateTime**  
  Representa uma data/hora com informações de fuso horário. Por exemplo, 2015-08-30T20:05:12.463-05:00[America/Mexico\_City].
* **OffsetDateTime**  
  Representa uma data/hora com um deslocamento de UTC/Greenwich. Por exemplo, 2015-08-30T20:05:12.463-05:00.
* **OffsetTime**  
  Representa uma hora com um deslocamento de UTC/Greenwich. Por exemplo, 20:05:12.463-05:00.

Assim como as classes do capítulo anterior, essas estão localizadas no pacote java.time e são imutáveis.

*(continua na próxima resposta)*

**Classes ZoneId e ZoneOffset**

O mundo está dividido em fusos horários nos quais o mesmo horário padrão é mantido. Por convenção, um fuso horário é expresso como o número de horas de diferença em relação ao Tempo Universal Coordenado (UTC). Como o Horário Médio de Greenwich (GMT) e o Tempo Zulu (Z), usado no meio militar, não têm deslocamento em relação ao UTC, eles são frequentemente usados como sinônimos.

O Java usa o banco de dados de fusos horários da **IANA** (Internet Assigned Numbers Authority), que mantém um registro de todos os fusos horários conhecidos ao redor do mundo e é atualizado muitas vezes por ano.

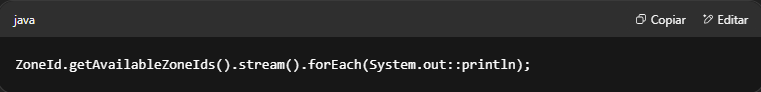
Cada fuso horário tem um ID, representado pela classe java.time.ZoneId. Existem três tipos de ID:

1. O primeiro tipo apenas declara o deslocamento em relação ao tempo UTC/GMT. Eles são representados pela classe ZoneOffset e consistem em dígitos começando com + ou -, por exemplo, +02:00.
2. O segundo tipo também declara o deslocamento em relação ao tempo UTC/GMT, mas com um dos seguintes prefixos: UTC, GMT e UT, por exemplo, UTC+11:00. Eles também são representados pela classe ZoneOffset.
3. O terceiro tipo é baseado em região. Esses IDs têm o formato área/cidade, por exemplo, Europe/London.

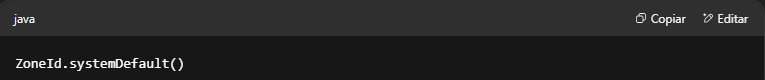
Você pode obter todos os IDs de zona disponíveis com o método estático:



Por exemplo, para imprimi-los no console:

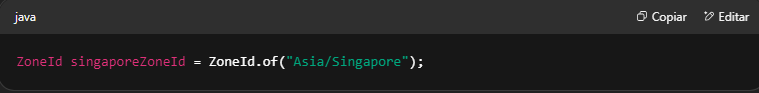


Para obter o ID de zona do seu sistema, use o método estático:



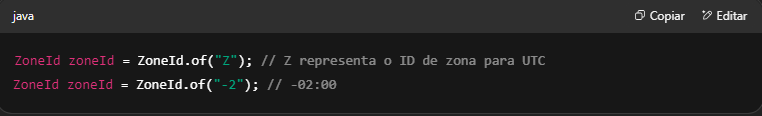
Internamente, ele usa java.util.TimeZone.getDefault() para encontrar o fuso horário padrão e o converte para um ZoneId.

Se você quiser criar um objeto ZoneId específico, use o método of():



Esse método analisa o ID produzindo um ZoneOffset ou um ZoneRegion (ambos estendem ZoneId).

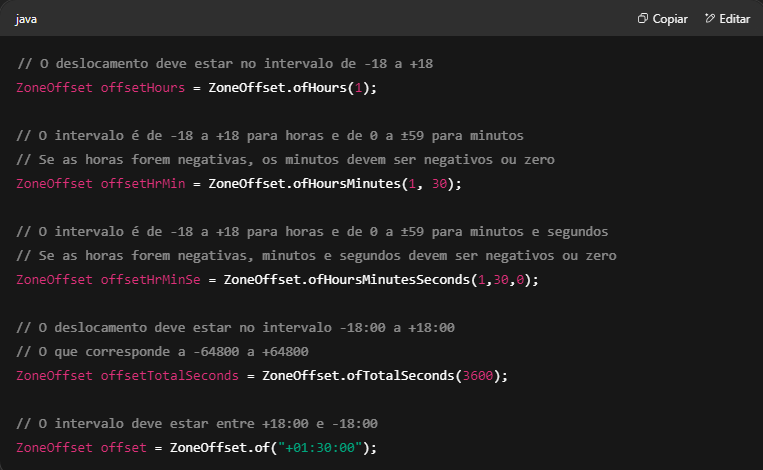
Na verdade, a linha acima produz um ZoneRegion. Um ZoneOffset é retornado se, por exemplo, o ID for Z, ou começar com + ou -. Por exemplo:



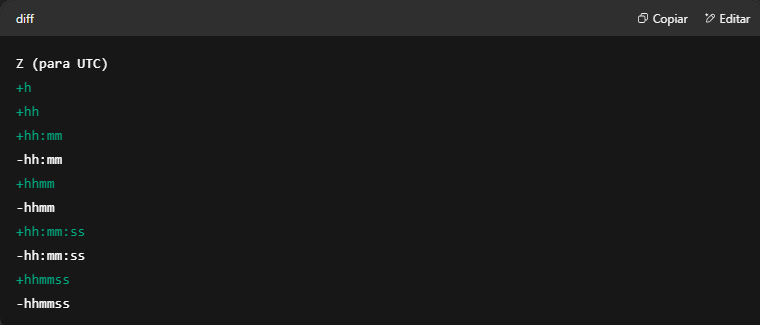
As regras para esse método são:

* Se o ID de zona for igual a Z, o resultado é ZoneOffset.UTC. Qualquer outra letra lançará uma exceção.
* Se o ID de zona começar com + ou -, o ID será interpretado como um ZoneOffset usando ZoneOffset.of(String).
* Se o ID de zona for igual a GMT, UTC ou UT, então o resultado é um ZoneId com o mesmo ID e regras equivalentes a ZoneOffset.UTC.
* Se o ID de zona começar com UTC+, UTC-, GMT+, GMT-, UT+ ou UT-, então o ID é dividido em dois, com um prefixo de duas ou três letras e um sufixo começando com o sinal. O sufixo é analisado como um ZoneOffset. O resultado será um ZoneId com o prefixo especificado e o ID de deslocamento normalizado.
* Todos os outros IDs são interpretados como IDs de zona baseados em região. Se o formato for inválido (ele deve coincidir com a expressão [A-Za-z][A-Za-z0-9~/.\_+-]+) ou não for encontrado, uma exceção será lançada.

Lembre-se de que um ZoneOffset representa um deslocamento, geralmente em relação ao UTC. Esta classe possui muito mais construtores do que ZoneId:

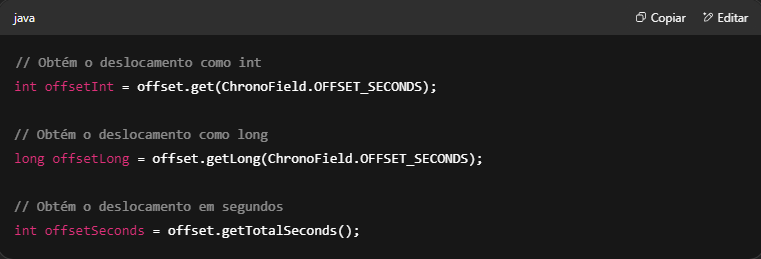


Os formatos aceitos pelo método of() são:



Se você passar um formato inválido ou um valor fora do intervalo para qualquer um desses métodos, uma exceção será lançada.

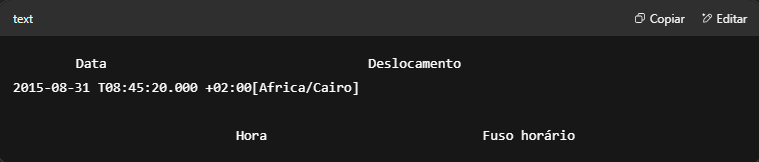
Para obter o valor do deslocamento, você pode usar:



ChronoField.OFFSET\_SECONDS é o único valor aceito de ChronoField, então as três instruções acima retornam o mesmo resultado. Outros valores lançam uma exceção.

De qualquer forma, uma vez que você tenha um objeto ZoneId, pode usá-lo para criar uma instância de ZonedDateTime.

**Um objeto ZonedDateTime**



**Classe ZonedDateTime**

Um objeto java.time.ZonedDateTime representa um ponto no tempo relativo a um fuso horário.

Um objeto ZonedDateTime tem três partes:

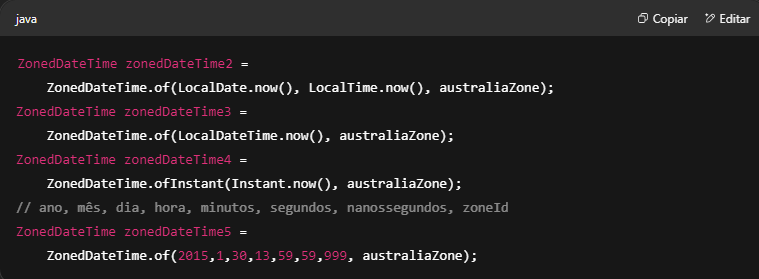
* Uma data
* Uma hora
* Um fuso horário

Isso significa que ele armazena todos os campos de data e hora, com precisão de nanossegundos, e um fuso horário com um deslocamento de zona.

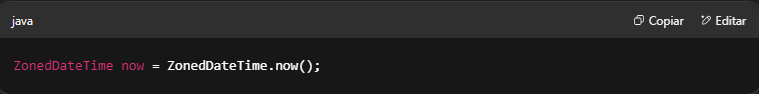
Assim, uma vez que você tenha um objeto ZoneId, pode combiná-lo com um LocalDate, LocalDateTime ou Instant, para transformá-lo em um ZonedDateTime:



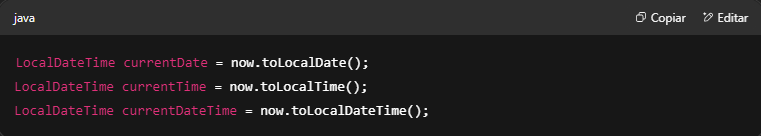
Ou usando o método of:



Você também pode obter a data/hora atual do relógio do sistema no fuso horário padrão com:



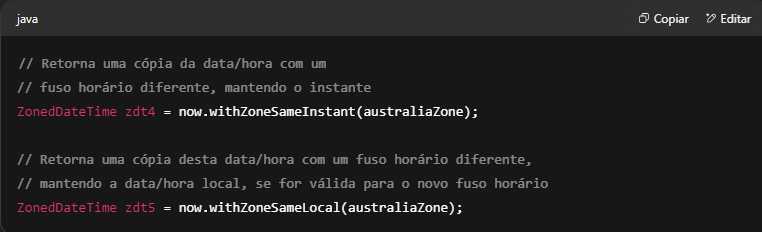
A partir de um ZonedDateTime, você pode obter LocalDate, LocalTime ou LocalDateTime (sem a parte do fuso horário) com:



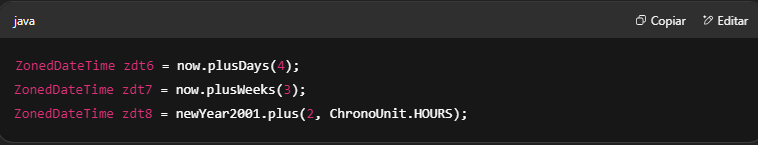
ZonedDateTime também possui a maioria dos métodos de LocalDateTime que vimos no capítulo anterior:



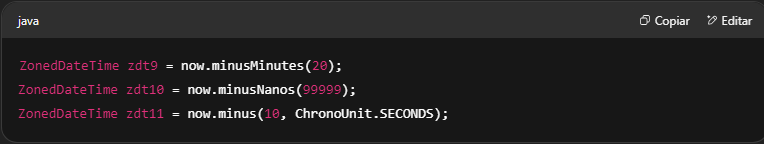
Os dois métodos a seguir são específicos de ZonedDateTime:



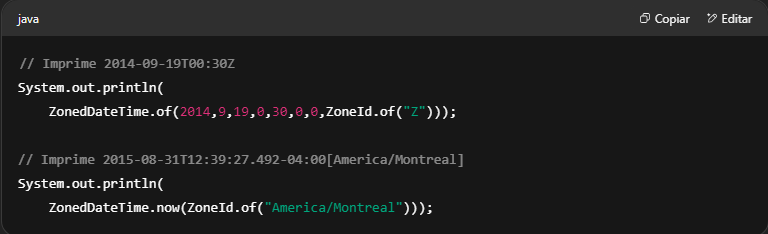
Adição



Subtração



O método toString() retorna a data/hora no formato de um LocalDateTime seguido por um ZoneOffset, opcionalmente um ZoneId se ele for diferente do offset, e omite as partes com valor zero:



**Horário de verão**

Muitos países no mundo adotam o que é chamado de Horário de Verão (Daylight Saving Time - DST), a prática de adiantar o relógio em uma hora no verão (bem, não exatamente no verão em todos os países, mas vamos seguir com isso) quando o horário de verão começa.

Quando o horário de verão termina, os relógios são atrasados em uma hora. Isso é feito para fazer melhor uso da luz natural do dia.

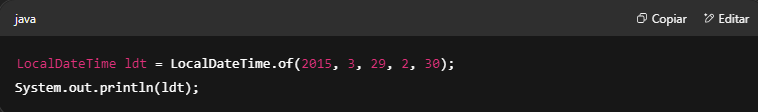
ZonedDateTime está totalmente ciente do horário de verão (DST).

Por exemplo, vamos considerar um país onde o DST é totalmente observado, como a Itália (UTC/GMT +2).

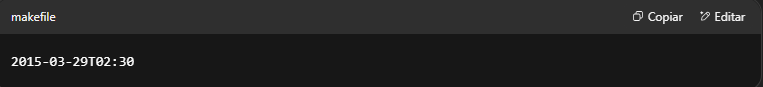
Em 2015, o DST começou na Itália em 29 de março e terminou em 25 de outubro. Isso significa que em:

* 29 de março de 2015 às 02:00:00 da manhã os relógios foram adiantados 1 hora para  
  29 de março de 2015 às 03:00:00 da manhã, horário local de verão.  
  *(Portanto, um horário como 29 de março de 2015 às 02:30:00 da manhã realmente não existiu!)*
* 25 de outubro de 2015 às 03:00:00 da manhã os relógios foram atrasados 1 hora para  
  25 de outubro de 2015 às 02:00:00 da manhã, horário local de verão.  
  *(Portanto, um horário como 25 de outubro de 2015 às 02:30:00 da manhã existiu duas vezes!)*

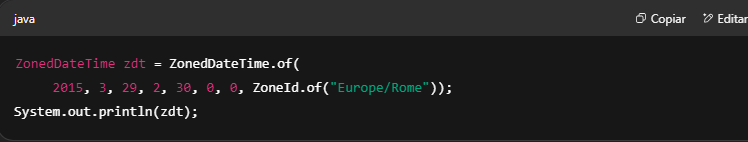
Se criarmos uma instância de LocalDateTime com essa data/hora e a imprimirmos:



O resultado será:



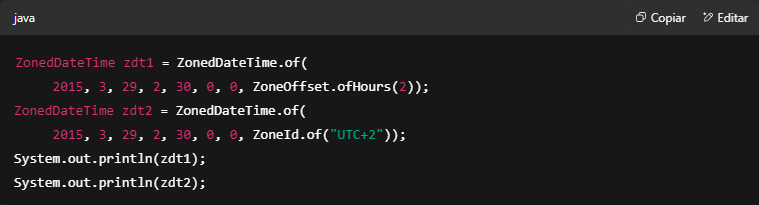
Mas se criarmos uma instância de ZonedDateTime para a Itália (note que o formato usa uma cidade, não um país) e imprimirmos:



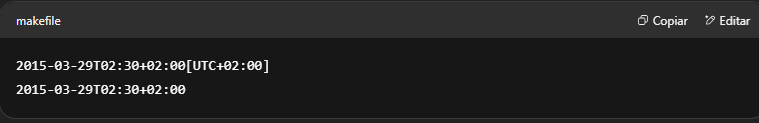
O resultado será exatamente como no mundo real ao usar DST:



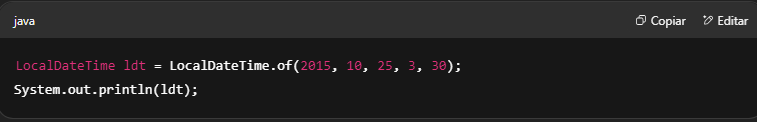
Mas cuidado. Temos que usar um ZoneId regional — um ZoneOffset não fará o trabalho, pois essa classe não possui as informações de regras de zona necessárias para considerar o horário de verão:



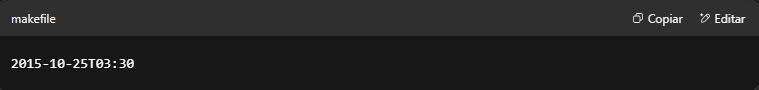
O resultado será:



Quando o DST termina, algo semelhante acontece:



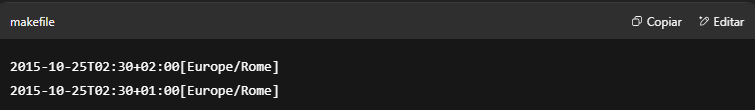
O resultado será:



Ao criarmos uma instância de ZonedDateTime para a Itália, devemos adicionar uma hora para ver o efeito (caso contrário, estaremos criando o ZonedDateTime às 03:00 do novo horário):

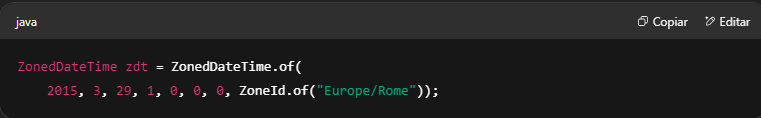


O resultado será:

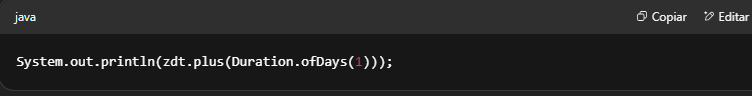


Também precisamos ter cuidado ao ajustar o horário através da fronteira do horário de verão usando uma versão dos métodos plus() e minus() que recebe uma implementação de TemporalAmount, em outras palavras, um Period ou uma Duration. Isso porque ambos diferem em seu tratamento do horário de verão.

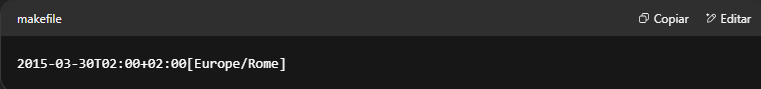
Considere uma hora antes do início do DST na Itália:



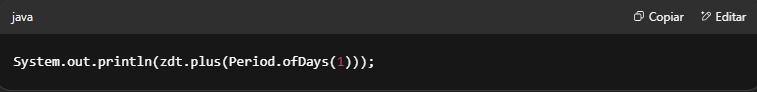
Quando adicionamos uma Duration de um dia:



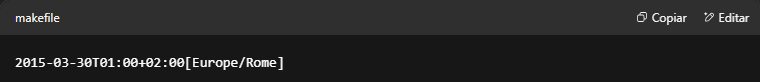
O resultado é:



Quando adicionamos um Period de um dia:



O resultado é:



O motivo é que Period adiciona um dia conceitual, enquanto Duration adiciona exatamente um dia (24 horas ou 86.400 segundos), e quando atravessa a fronteira do horário de verão, uma hora é adicionada, e o horário final é 02:00 em vez de 01:00.

**OffsetDateTime e OffsetTime**

OffsetDateTime representa um objeto com informações de data/hora e um deslocamento em relação ao UTC, por exemplo:



Você pode pensar que Instant, OffsetDateTime e ZonedDateTime são muito parecidos — afinal, todos armazenam data e hora com precisão de nanossegundos. No entanto, há diferenças sutis mas importantes:

* Instant representa um ponto no tempo no fuso horário UTC.
* OffsetDateTime representa um ponto no tempo com um deslocamento (qualquer deslocamento).
* ZonedDateTime representa um ponto no tempo em um fuso horário (qualquer fuso horário), acrescentando regras completas de fuso, como ajustes de horário de verão.

OffsetTime representa uma hora com um deslocamento do UTC, por exemplo, 11:30-06:00. A maneira comum de criar instâncias dessas classes é:

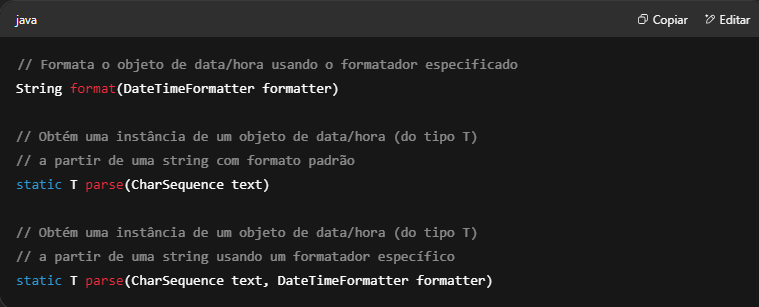


Ambas as classes possuem praticamente os mesmos métodos que suas contrapartes LocalDateTime, ZonedDateTime e LocalTime. Com um deslocamento do UTC e sem variações de fuso horário, elas sempre representam um instante exato no tempo, o que pode ser mais adequado para certos tipos de aplicações (a documentação do Java recomenda OffsetDateTime ao se comunicar com banco de dados ou em protocolos de rede).

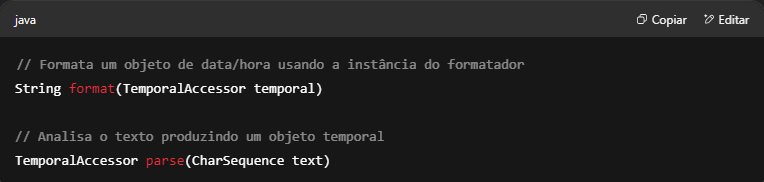
**Análise e Formatação**

java.time.format.DateTimeFormatter é a nova classe para analisar (parse) e formatar datas. Ela pode ser usada de duas formas:

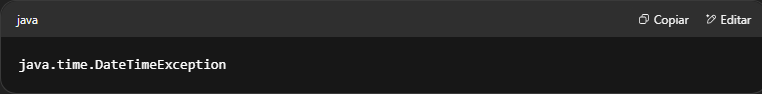
As classes de data/hora LocalDate, LocalTime, LocalDateTime, ZonedDateTime, OffsetDate e OffsetDateTime todas possuem os seguintes três métodos:



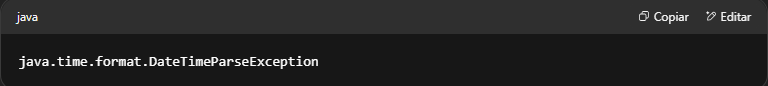
DateTimeFormatter possui os seguintes dois métodos:



Todos os métodos de formatação lançam a exceção em tempo de execução:



Todos os métodos de análise (parse) lançam a exceção em tempo de execução:



DateTimeFormatter fornece três maneiras de formatar objetos de data/hora:

* **Formatadores pré-definidos**
* **Formatadores específicos de localidade (locale)**
* **Formatadores com padrões personalizados**

**Formatadores pré-definidos**

| **Formatador** | **Descrição** | **Exemplo** |
| --- | --- | --- |
| BASIC\_ISO\_DATE | Campos de data sem separadores | 20150803 |
| ISO\_LOCAL\_DATE | Campos de data com separadores | 2015-08-03 |
| ISO\_LOCAL\_TIME | Campos de hora com separadores | 13:40:10 |
| ISO\_LOCAL\_DATE\_TIME | Data e hora | 2015-08-03T13:40:10 |
| ISO\_OFFSET\_DATE | Data com deslocamento | 2015-08-03+07:00 |
| ISO\_OFFSET\_TIME | Hora com deslocamento | 13:40:10+07:00 |
| ISO\_OFFSET\_DATE\_TIME | Data e hora com deslocamento | 2015-08-03T13:40:10+07:00 |
| ISO\_ZONED\_DATE\_TIME | Data e hora com fuso | 2015-08-03T13:40:10+07:00[Asia/Bangkok] |
| ISO\_DATE, ISO\_TIME | Data ou hora com ou sem deslocamento | 2015-08-03+07:00, 13:40:10 |
| ISO\_DATE\_TIME | Data/hora com ZoneId | 2015-08-03T13:40:10+07:00[Asia/Bangkok] |
| ISO\_INSTANT | Instante UTC | 2015-08-03T13:40:10Z |
| ISO\_ORDINAL\_DATE | Ano e dia do ano | 2015-200 |
| ISO\_WEEK\_DATE | Ano, semana e dia da semana | 2015-W34-2 |
| RFC\_1123\_DATE\_TIME | Formato de data RFC 1123 / RFC 822 | Mon, 3 Ago 2015 13:40:10 GMT |

**Estilos Localizados**

Você também pode usar estilos localizados para formatar datas e horários. Os estilos disponíveis são:

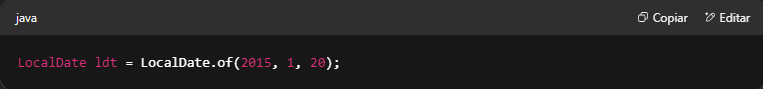
| **Estilo** | **Data** | **Hora** |
| --- | --- | --- |
| SHORT | 8/3/15 | 1:40 PM |
| MEDIUM | Aug 03, 2015 | 1:40:00 PM |
| LONG | August 03, 2015 | 1:40:00 PM PDT |
| FULL | Monday, August 03, 2015 | 1:40:00 PM PDT |

**Símbolos de Padrões Personalizados**

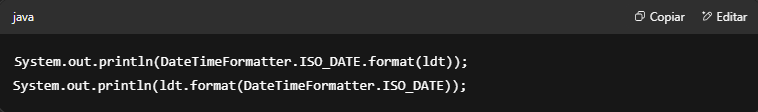
| **Símbolo** | **Significado** | **Exemplos** |
| --- | --- | --- |
| G | Era | AD; Anno Domini; A |
| u | Ano | 2015; 15 |
| y | Ano da era | 2015; 15 |
| D | Dia do ano | 150 |
| M / L | Mês do ano | 7; 07; Jul; July; J |
| d | Dia do mês | 20 |
| Q / q | Trimestre do ano | 2; 02; Q2; 2nd quarter |
| Y | Ano baseado em semana | 2015; 15 |
| w | Semana do ano (baseado em semana) | 30 |
| W | Semana do mês | 2 |
| E | Dia da semana | Tue; Tuesday; T |
| e / c | Dia da semana local | 2; 02; Tue; Tuesday; T |
| F | Ocorrência do dia na semana no mês | 2 |
| a | AM/PM do dia | AM |
| h | Hora (1–12) | 10 |
| K | Hora (0–11) | 1 |
| k | Hora (1–24) | 20 |
| H | Hora (0–23) | 23 |
| m | Minuto | 10 |
| s | Segundo | 11 |
| S | Fração de segundo | 999 |
| A | Milissegundos do dia | 2345 |
| n | Nanossegundos do segundo | 865437987 |
| N | Nanossegundos do dia | 12986497300 |
| V | ID do fuso horário | Asia/Manila; Z; -06:00 |
| z | Nome do fuso horário | Pacific Standard Time; PST |
| O | Deslocamento localizado | GMT+4; GMT+04:00; UTC-04:00 |
| X | Deslocamento de zona ("Z" para zero) | Z; -08; -0830; -08:30 |
| x | Deslocamento de zona | +0000; -08; -0830; -08:30 |
| Z | Deslocamento de zona | +0000; -0800; -08:00 |
| ' | Escape para texto | 'GMT' → GMT |
| '' | Aspas simples | '' → ' |
| [] | Seção opcional (início/fim) |  |
| {} | Reservado para uso futuro |  |

**Exemplos de Padrões Personalizados**

Supondo:



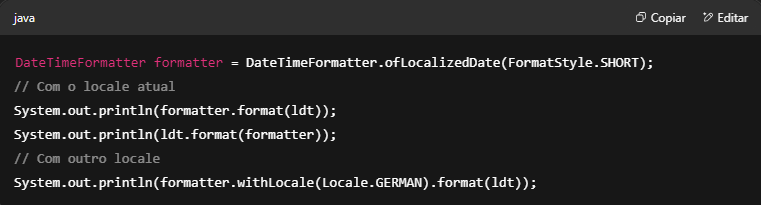
Exemplos usando um formatador pré-definido:



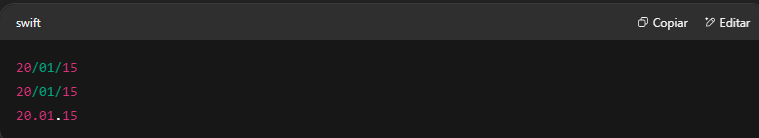
Saída:



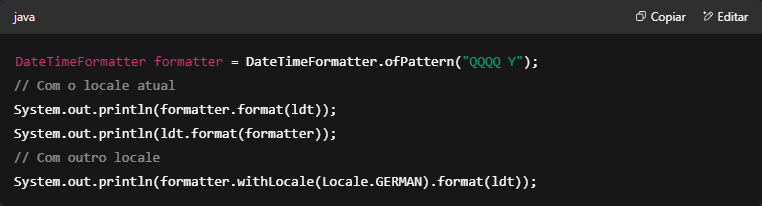
Exemplos usando um estilo localizado:



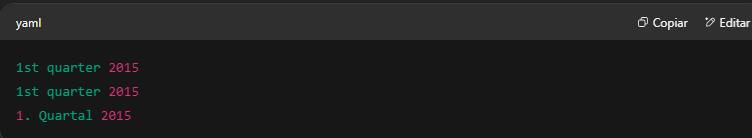
Saída possível:



Exemplos com um padrão personalizado:



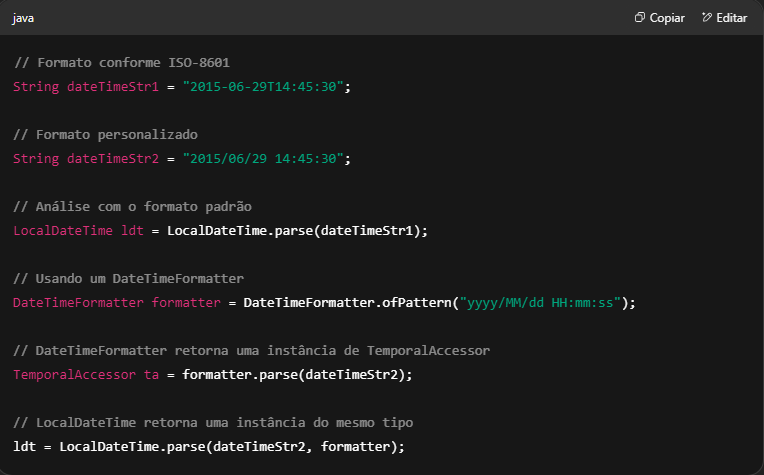
Saída possível:



Se o formatador usar informações que não estão disponíveis, uma DateTimeException será lançada. Por exemplo, ao usar DateTimeFormatter.ISO\_OFFSET\_DATE com uma instância de LocalDate (que não possui informações de deslocamento).

**Análise (Parse) de valores de data/hora a partir de uma string**

Para analisar um valor de data e/ou hora a partir de uma string, use um dos métodos parse(). Por exemplo:



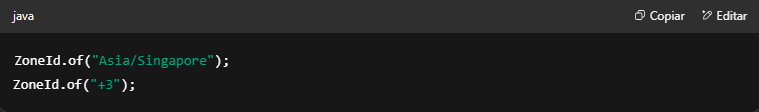
A versão de parse() das classes de data/hora aceita uma string em formato conforme a norma ISO-8601. Veja a tabela:

| **Classe** | **Formato ISO-8601** | **Exemplo** |
| --- | --- | --- |
| LocalDate | uuuu-MM-dd | 2007-12-03 |
| LocalTime | HH:mm:ss | 10:15:30 |
| LocalDateTime | uuuu-MM-dd'T'HH:mm:ss | 2007-12-03T10:15:30 |
| ZonedDateTime | uuuu-MM-dd'T'HH:mm:ssXXXXX[VV] | 2011-12-03T10:15:30+01:00[Europe/Paris] |
| OffsetDateTime | uuuu-MM-dd'T'HH:mm:ssXXXXX | 2011-12-03T10:15:30+01:00 |
| OffsetTime | HH:mm:ssXXXXX | 10:15:30+01:00 |

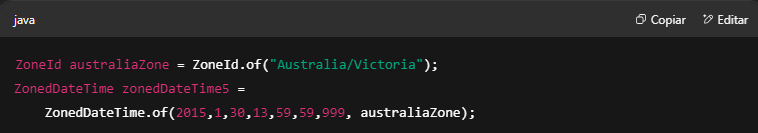
Se o formatador utilizar informações não disponíveis, ou se o padrão do formato for inválido, uma DateTimeParseException será lançada.

**Pontos-Chave**

* ZoneId, ZoneOffset, ZonedDateTime, OffsetDateTime e OffsetTime são as classes da nova API de Data/Hora do Java que armazenam informações sobre fusos horários e deslocamentos. Elas estão localizadas no pacote java.time e são imutáveis.
* Cada fuso horário possui um ID, representado pela classe ZoneId. Existem três tipos:
  1. Apenas deslocamento, como +02:00 (ZoneOffset)
  2. Deslocamento com prefixo UTC, GMT ou UT, como UTC+11:00 (ZoneOffset)
  3. Baseado em região, como Europe/London
* Para criar um objeto ZoneId específico, use:



* Os dois primeiros produzem um ZoneRegion. O último, um ZoneOffset.
* Um objeto ZonedDateTime representa um ponto no tempo relativo a um fuso horário. Pode ser criado com:



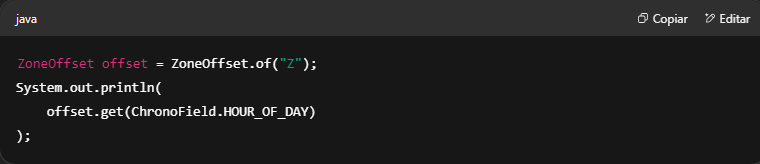
* Ou usando LocalDate, LocalTime, LocalDateTime ou Instant + ZoneId.
* ZonedDateTime respeita o horário de verão, ajustando a hora automaticamente quando ele começa ou termina.
* Period e Duration tratam o horário de verão de forma diferente:
* Period adiciona um "dia conceitual"
* Duration adiciona exatamente 24 horas
* OffsetDateTime representa um ponto no tempo com deslocamento em relação ao UTC, por exemplo, 2015-01-01T11:30-06:00.
* OffsetTime representa uma hora com deslocamento do UTC, como 11:30-06:00.
* DateTimeFormatter é a nova classe para formatação/análise de datas, usada com:
* format(DateTimeFormatter)
* parse(CharSequence)
* parse(CharSequence, DateTimeFormatter)
* Métodos de format() lançam DateTimeException; métodos de parse() lançam DateTimeParseException.

**Autoavaliação (Self Test)**

**1. Quais das seguintes são formas válidas de criar um objeto ZoneId?**

A. ZoneId.ofHours(2);  
B. ZoneId.of("2");  
C. ZoneId.of("-1");  
D. ZoneId.of("America/Canada");

**2. Dado o código:**



Qual dos seguintes é o resultado da execução das linhas acima?

A. 0  
B. 1  
C. 12:00  
D. Uma exceção é lançada

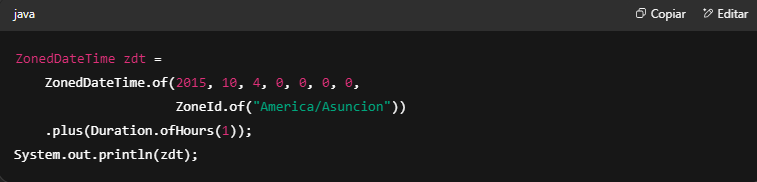
**3. Dado:**



Assumindo um fuso horário local de +02:00, qual das seguintes é a saída?

A. 05:00  
B. 17:00  
C. 02:00  
D. 03:00

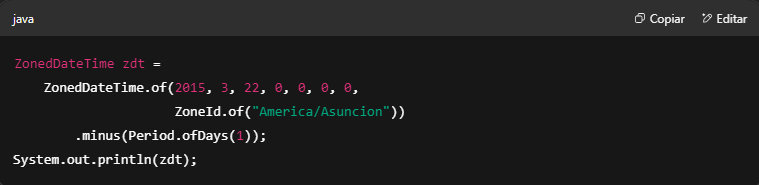
**4. Dado:**



Assumindo que o horário de verão começa em 4 de outubro de 2015 às 00:00:00, qual é o resultado da execução das linhas acima?

A. 2015-10-04T00:00-03:00[America/Asuncion]  
B. 2015-10-04T01:00-03:00[America/Asuncion]  
C. 2015-10-04T02:00-03:00[America/Asuncion]  
D. 2015-10-03T23:00-03:00[America/Asuncion]

**5. Dado:**



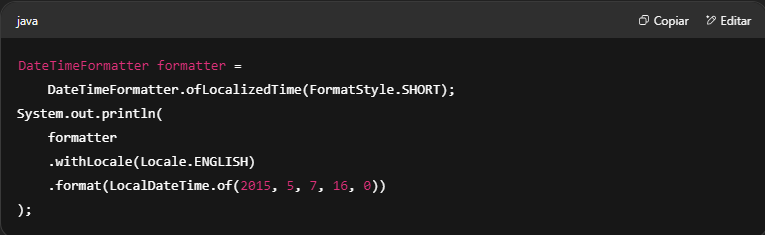
Assumindo que o horário de verão termina em 22 de março de 2015 às 00:00:00, qual é o resultado?

A. 2015-03-21T01:00-03:00[America/Asuncion]  
B. 2015-03-21T00:00-03:00[America/Asuncion]  
C. 2015-03-20T23:00-03:00[America/Asuncion]  
D. 2015-03-21T02:00-03:00[America/Asuncion]

**6. Quais das seguintes afirmações são verdadeiras?**

A. java.time.ZoneOffset é uma subclasse de java.time.ZoneId.  
B. java.time.Instant pode ser obtido a partir de java.time.ZonedDateTime.  
C. java.time.ZoneOffset consegue gerenciar DST.  
D. java.time.OffsetDateTime representa um ponto no tempo no fuso horário UTC.

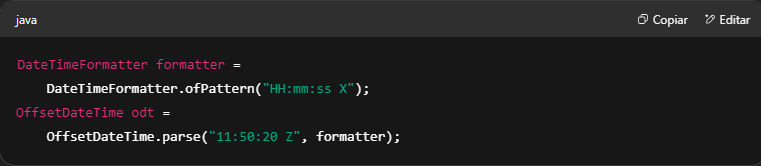
**7. Dado:**



Qual das seguintes é a saída?

A. 5/7/15 4:00 PM  
B. 5/7/15  
C. 4:00 PM  
D. 4:00:00 PM

**8. Dado:**



Qual das seguintes afirmações é verdadeira?

A. O padrão "HH:mm:ss X" é inválido.  
B. Um OffsetDateTime é criado com sucesso.  
C. Z é um deslocamento inválido.  
D. Uma exceção é lançada em tempo de execução.