

Data, Hora e Instante

2.5 - Java Time

i Informação

Antes de mergulhar nestes novos recursos da linguagem certifique-se de ter conhecimentos sobre os conceitos básicos de trabalhar com Data e Hora em Java.

2.5.1 - Introdução

Trabalhar com o tempo ou o instante atual é um dos requisitos mais complexos na área da programação onde exige uma compreensão no que se refere a criação, manipulação, comparação e formatação de data e hora.



Aprendemos que a linguagem é composta para um conjunto de classes e métodos que representam datas e horas em um programa, mas a partir do Java 8 foi incorporado de forma nativa os recursos até então disponibilizados através de um projeto denominado de **Joda Time**.

Assim, hoje a linguagem é composta por classes que representam respectivamente:

- **Data**

- Hora
- Data e Hora
- Instante

Sucesso

Todas as operações relacionadas a data e hora retornam a um objeto (valor) imutável possibilitando maior segurança quanto as manipulações necessárias.

2.5.2 - LocalDate

O LocalDate representa uma data no formato ISO (aaaa-MM-dd) sem hora. Podemos usá-lo para armazenar datas como aniversários e dias de pagamento.



2.5.2.1 Instanciação

Já aprendemos que instanciar refere-se a criar um novo objeto onde o meio mais tradicional é pelo uso de **Construtores**, porém as classes citadas acima possuem

métodos estáticos de criação de objetos correspondentemente.

Uma instância da data atual pode ser criada a partir do relógio do sistema:

```
1  import java.time.LocalDate;
2
3  public class JavaTime {
4      public static void main(String[] args) {
5
6          LocalDate dataAtual = LocalDate.now();
7
8          System.out.println(dataAtual);
9          //Representação de um LocalDate: 2023-03-01
10     }
11 }
```

java

Também é possível determinar os dígitos para dia, mês e ano em um `LocalDate` utilizando o método `of (year, month, dayOfMonth)` .

```
1  LocalDate dataEspecific1 = LocalDate.of(2023,1,12);
2
3  LocalDate dataEspecific2 = LocalDate.of(2023, Month.JANUARY,12);
```

java

Atenção

Dois itens devem ser analisados no exemplo acima:

Primeiro: Java Time considera que meses começam com 1 (janeiro), 2 (fevereiro) e ...

Segundo: Existe um enum denominado `Month` com os valores dos meses pré-definidos

Agora imagina que você tenha uma `String` onde seu conteúdo representa uma data, onde você gostaria de converter para um `LocalDate` correspondente? Simples, veja o código abaixo:

```
1  // este conteúdo pode ser um argumento de seu programa via console
2  // ou toString() de algum componente visual
3  String stringData = "2023-01-17";
4
5  LocalDate dataConcreta = LocalDate.parse(stringData);
```

java

Até aqui já aprendemos variedades sobre Data utilizando Java Time, mas nós como bons brasileiros sabemos que o formato de uma data informada no sistemas desktops, web e aplicativos possuem um formato diferente. Então como converteríamos uma string de data vindo no formato `dd/mm/aaaa` ? Este é um trabalho para a classe `java.time.format.DateTimeFormatter`

```
1 String stringDataBr = "17/01/2023";  
2  
3 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");  
4  
5 LocalDate dataConcreta = LocalDate.parse(stringDataBr, formatter);
```

java

Cuidado

Tentar converter uma String em um JavaTime que não tenha uma compatibilidade de conversão (parse) gerá uma exceção semelhante a esta:

```
java.time.format.DateTimeParseException: Text '17/01/23' could not be parsed
```

2.5.2.2 Manipulação

Depois de um objeto `LocalDate` criado na sua aplicação, você poderá, manipulá-lo resultando na criação de novos objetos com os valores resultantes de operações como: adicionar dia, subtrair mês e etc.

Data Especial Quando fará 15 anos?

```
1 LocalDate aniversarioIzabelly = LocalDate.of(2023, 5, 3);
```

java

Sucesso

Agora que já compreendeu grande parte dos recursos de um `LocalDate`, que tal explorar os demais métodos `plusXX` e `minusYY` ?

2.5.2.3 Comparação

Outro aspecto extremamente relevante quando trabalhamos com data é realizar comparações entre duas datas com base no: ano, mes, dia, semana e etc.

java

```
1  LocalDate data1 = LocalDate.of(2023,3,4);
2  LocalDate data2 = LocalDate.of(2024,4,3);
3
4  //data2 está após a data 1? true
5  System.out.println(data2.isAfter(data1));
6
7  //data2 está antes que data 1? false
8  System.out.println(data2.isBefore(data1));
9
10 //data2 é igual a data 1? false
11 System.out.println(data2.isEqual(data1));
```

2.5.2.4 Formatação

Assim como explanamos formatação de datas utilizando DateFormat e SimpleDateFormat, agora o Java Time disponibiliza recursos para formatações de data e hora nesta nova abordagem da linguagem.

Abaixo iremos explorar formatações considerando um formato pré-estabelecido e em seguida formatos com base na Local e (idioma) configurado em sua aplicação.

java

```
1  LocalDate data = LocalDate.of(2023,1,13);
2
3  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
4
5  System.out.println(data); //2023-01-13
6
7  //formatando a data criada acima
8  System.out.println(formatter.format(data)); //13/01/2023
```

Se você pretende desenvolver uma aplicação no qual é necessário considerar a região em que o usuário acessará o sistema para formatar campos data adequadamente, é possível realizar esta formatação com quatro formatos pré-definidos:

java

```

1  LocalDate data = LocalDate.of(2023,1,13);
2
3  String dataFormatada = data.format(DateTimeFormatter.ofLocalizedDate(FormatStyle
4  dataFormatada = data.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM
5  dataFormatada = data.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT)
6  dataFormatada = data.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL))

```

🔔 Atenção

A formatação apresentada acima poderá ter um comportamento diferente considerando as configurações de região e idioma em sistema operacional.

Para definir uma formatação mediante uma seleção de um idioma em sua aplicação, recomendo explorar um pouco mais sobre o uso de Locale

Considerando a imagem abaixo, vamos apresentar a mesma data 23/08/2021 nas respectivas formatações abaixo:



java

```

1
2  LocalDate data = LocalDate.of(2021,8,23);
3
4  Locale[] locales = {Locale.CANADA, Locale.US, new Locale("fi","FI"),Locale.UK};
5
6  for(Locale locale:locales){
7      String dataFormatada = data.format(DateTimeFormatter.ofLocalizedDate(FormatS
8
9

```

```
8      System.out.println(dataFormatada);
9  }
10
11  // 2021-08-23
12  // 8/23/21
13  // 23.8.2021
14  // 23/08/2021
```

2.5.2.5 Dados específicos

Depois de aprendermos a intanciar, manipular e formatar datas utilizando os recursos do Java Time, chegou a hora de obter alguns dados isoladamente através de métodos específicos:

```
1  LocalDate data = LocalDate.of(2021,8,23);
2
3  System.out.println(data.getDayOfMonth()); // retorna o dia do mês -> 23
4  System.out.println(data.getYear());       // retorna o ano -> 2023
5  System.out.println(data.getMonth());      // retorna o elemento de enum java.t
6  System.out.println(data.getMonthValue()); // retorna o número do mês entre 1-12
```

java

2.5.3 - LocalTime

O LocalTime representa a hora sem data.

Semelhante a LocalDate, podemos criar uma instância de LocalTime a partir do relógio do sistema ou usando os métodos parse e of.

Vamos agora dar uma olhada rápida em algumas das APIs comumente usadas.

Uma instância do LocalTime atual pode ser criada a partir do relógio do sistema:



2.5.3.1 Instanciação

java

```
1  LocalTime horaAtual = LocalTime.now();
2
3  LocalTime horaMinutoSegundoEspecifico = LocalTime.of(13,21,42); // 13:21:42
4
5  LocalTime horaMinutoEspecifico = LocalTime.of(13,21); // 13:21
6
7  LocalTime horaDeUmaString = LocalTime.parse("13:21:42");
8
9  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HHmmss");
10 LocalTime horaDeUmaStringDespadronizada = LocalTime.parse("132142", formatter);
```

Atenção

Memorize ao máximo esta tabelinha quando quiser converter / formatar data e hora com estruturas específicas:

Letra	Campo	Exemplo
G	Era designator	AD
y	Year	2018 (yyyy), 18 (yy)
M	Month in year	July (MMMM), Jul (MMM), 07 (MM)

Letra	Campo	Exemplo
w	Results in week in year	16
W	Results in week in month	3
D	Gives the day count in the year	266
d	Day of the month	09 (dd), 9(d)
F	Day of the week in month	4
E	Day name in the week	Tuesday, Tue
u	Day number of week	where 1 represents Monday, 2 represents Tuesday and so on 2
a	AM or PM marker	AM
H	Hour in the day	(0-23) 12
k	Hour in the day	(1-24) 23
K	Hour in am/pm	for 12 hour format (0-11) 0
h	Hour in am/pm	for 12 hour format (1-12) 12
m	Minute in the hour	59
s	Second in the minute	35
S	Millisecond in the minute	978
z	Timezone Pacific Standard	Time; PST; GMT-08:00
Z	Timezone offset in hours (RFC pattern)	-0800
X	Timezone offset in ISO format	-08; -0800; -08:00

2.5.3.2 Manipulação

java

```
1    LocalDateTime horaAtual = LocalDateTime.now();
2    LocalDateTime horaAtualMais30Minutos = horaAtual.plusMinutes(30);
3    LocalDateTime horaAtualMenos2Horas = horaAtual.minusHours(2);
4
5    System.out.println(horaAtual);
6    System.out.println(horaAtualMais30Minutos);
7    System.out.println(horaAtualMenos2Horas);
```

2.5.3.3 Comparação

java

```
1    LocalDateTime horaAtual = LocalDateTime.now();
2    LocalDateTime horaAtualMenos2Horas = horaAtual.minusHours(2);
3
4    boolean passado = horaAtual.isBefore(horaAtualMenos2Horas);
5
6    System.out.println(passado);
```

2.5.3.4 Formatação

java

```
1    LocalDateTime hora = LocalDateTime.of(13, 21, 42);
2    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("ss-mm-HH");
3    String horaFormatada = hora.format(formatter);
4    System.out.println(horaFormatada); //42-21-13
```

2.5.3.5 Dados específicos

java

```
1    LocalDateTime hora = LocalDateTime.of(13, 21, 42);
2
3    System.out.println(hora.getHour());    //13
4    System.out.println(hora.getMinute()); //21
5    System.out.println(hora.getSecond()); //42
6    System.out.println(hora.getNano());   //0
```

2.5.4 - LocalDateTime

LocalDateTime é usado para representar uma combinação de data e hora. Essa é a classe mais usada quando precisamos de uma combinação de data e hora.



2.5.4.1 Instanciação

java

```
1 //2023-06-25T16:25 -> data e hora sem segundo
2 LocalDateTime dataHora = LocalDateTime.of(2023,6,25, 16,25);
3
4 //2023-06-25T16:25:33 -> data e hora com segundo
5 dataHora = LocalDateTime.of(2023,6,25, 16,25,33);
```

2.5.4.2 Manipulação

java

```
1 LocalDateTime dataHora = LocalDateTime.of(2023,6,25, 16,25);
2
3 LocalDateTime proximoMes = dataHora.plusMonths(1);
4
5 LocalDateTime dataHoraComMenos15Minutos = dataHora.minusMinutes(15);
6
```

```
LocalDateTime dataHoraDefinindoSegundoZero = dataHora.withSecond(33);
```

2.5.4.3 Comparação

java

```
1  LocalDateTime dataHora1 = LocalDateTime.of(2023,6,25, 16,25);
2  LocalDateTime dataHora2 = LocalDateTime.of(2023,6,25, 16,25,17);
3
4  // comparando dois objetos data/hora em sua totalidade
5  boolean dataHoraIgual = dataHora1.equals(dataHora2); //false
6
7  //comparando dígito específico de uma data e hora
8  boolean dataHoraDigitoIgual = dataHora1.getDayOfMonth()==dataHora2.getDayOfMonth
```

2.5.4.4 Formatação

java

```
1  LocalDateTime dataHora = LocalDateTime.of(2023,7,22,17,33);
2  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yy HH:mm:ss");
3  String dataHoraFormatada = dataHora.format(formatter);
4  System.out.println(dataHoraFormatada); // 22/07/23 17:33:00
```

2.5.5 - Instant

A classe Java Instant é usada para representar um momento específico na linha do tempo. Isso pode ser usado para registrar carimbos de data/hora do evento no aplicativo. Esta classe é imutável e thread-safe.

Ao contrário do antigo `java.util.Date` que tem precisão de milissegundos, um `Instant` tem precisão de nanossegundos. Representar um ponto no tempo usando precisão de nanossegundos requer o armazenamento de um número maior que um longo. Para conseguir isso, a classe armazena um `long` que representa os segundos de época e um `int` que representa nanossegundos de segundo, que sempre estará entre 0 e 999.999.999. Os segundos da época são medidos a partir da época Java padrão de 1970-01-01T00:00:00Z, onde os instantes após a época têm valores positivos e os instantes anteriores têm valores negativos.



2.5.5.1 Instanciação

java

```
1
2 //1970-01-01T00:00:00Z até 2023-03-13T13:18:00Z
3 long milisegundos = 1678713480000L;
4 Instant instant = Instant.ofEpochMilli(milisegundos);
5 System.out.println(instant);
6
7 long segundos = milisegundos / 1000;
8 instant = Instant.ofEpochSecond(segundos);
9 System.out.println(instant);
10
```

2.5.5.2 Manipulação

java

```
1 Instant instant = Instant.parse("1984-08-13T10:15:30.345Z");
2
3 System.out.println("Instant          : " + instant);
4
5 // Adding/subtracting seconds
6 System.out.println("15 seconds before : " + instant.minusSeconds(15));
7 System.out.println("10 seconds after  : " + instant.plusSeconds(10));
```

```

8
9 // Adding/subtracting millis
10 System.out.println("Minus 45000 millis : " + instant.minusMillis(45000));
11 System.out.println("Plus 10000 millis : " + instant.plusMillis(10000));
12
13 // Adding/subtracting nanos
14 System.out.println("Minus 45123456 nanos: " + instant.minusNanos(45123456));
15 System.out.println("Plus 111234567 nanos: " + instant.plusNanos(111234567));
16
17 // Using MINUTES
18 System.out.println("45 minutes before : " + instant.minus(45, ChronoUnit.MINUT
19 // Using HOURS
20 System.out.println("3 hours before : " + instant.minus(3, ChronoUnit.HOURS)
21 // Using MILLIS also supported
22 System.out.println("30000 millis later : " + instant.plus(30000, ChronoUnit.MIL
23 // Using DAYS
24 System.out.println("10 days later : " + instant.plus(10, ChronoUnit.DAYS))
25
26 // Using TemporalAmount - Duration
27 System.out.println("10 seconds before : " + instant.minus(Duration.ofSeconds(1
28 // Using TemporalAmount - Period
29 System.out.println("5 days later : " + instant.plus(Period.ofDays(5)));

```

2.5.5.3 Comparação

java

```

1 Instant instant1 = Instant.parse("1984-08-13T10:15:30.345Z");
2 Instant instant2 = Instant.now();
3
4 System.out.println(instant1.isAfter(instant2)); //false
5 System.out.println(instant1.equals(instant2)); //false

```

2.5.5.4 Formatação

java

```

1 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy").withZone
2
3 Instant instant = Instant.parse("2022-02-15T18:35:24.00Z");
4 String formattedInstant = formatter.format(instant);
5
6 System.out.println(formattedInstant); //15.02.2022

```

2.5.5.5 Dados específicos

java

```
1 Instant instant = Instant.now();
2 System.out.println("Instant : " + instant);
3
4 long epochSecond = instant.getEpochSecond();
5 System.out.println("EpochSecond: " + epochSecond);
6
7 int nano = instant.getNano();
8 System.out.println("Nano : " + nano);
```

Sucesso

Aprendemos que trabalhar com data e hora na linguagem Java é uma tarefa que exige muita compreensão e entendimento de conceitos essenciais, porém, procure focar e explorar um pouco mais os recursos de **instanciação**, **manipulação**, **parse** e **formatação** para estar mais que preparado para os desafios do dia-a-dia.

2.5.6 - Period e Duration

2.5.6.1 Period

A classe `Period` usa as unidades ano, mês e dia para representar um período de tempo.

Podemos obter um objeto Período como a diferença entre duas datas usando o método `between()` :

java

```
1 LocalDate hoje = LocalDate.of(2023, 2, 20);
2 LocalDate natal = LocalDate.of(2023, 12, 25);
3
4 Period period = Period.between(hoje, natal);
5
6 System.out.println(period.getYears()); //anos
7 System.out.println(period.getMonths()); //meses
8 System.out.println(period.getDays()); //dias
```

2.5.6.2 Duration

A classe `Duration` representa um intervalo de tempo em segundos ou nanossegundos e é mais adequada para lidar com períodos de tempo menores, em casos que exigem mais precisão.

Podemos determinar a diferença entre dois instantes como um objeto `Duration` usando o método `between()` :

```
1      LocalDateTime inícioTreino = LocalDateTime.of(6,30,33);
2      LocalDateTime fimTreino = LocalDateTime.of(7,16,14);
3
4      Duration duration = Duration.between(inícioTreino, fimTreino);
5
6      System.out.println(duration.getSeconds()); //2741 segundos
7
8      //convertendo para um LocalDateTime com base na duração
9      LocalDateTime tempoAtividade = LocalDateTime.ofSecondOfDay(duration.getSeconds());
10     System.out.println(tempoAtividade); //00:45:41
```

java

2.5.7 - Transformação

Chegou o momento mais temido quando estamos trabalhando com data e hora em nossa aplicação, a transformação de objetos de acordo com requisitos apresentados ao longo do projeto.

Vamos explorar alguns recursos que o Java Time oferece em cada situação correspondentemente:

Informação

Lembre-se, a linguagem Java é muito rigorosa quando se refere a tipagem dos dados

2.5.7.1 LocalDateTime to LocalDate

```
1      LocalDateTime dataHora = LocalDateTime.of(2023, 1, 24, 16,31,27);
2
3      LocalDate data = dataHora.toLocalDate();
```

java

2.5.7.2 LocalDateTime to LocalDateTime

java

```
1 LocalDateTime dataHora = LocalDateTime.of(2023, 1, 24, 16,31,27);
2
3 LocalTime hora = dataHora.toLocalTime();
```

2.5.7.3 LocalDate to LocalDateTime

java

```
1 LocalDate data = LocalDate.of(2023, 1, 24);
2
3 LocalDateTime dataHora = data.atTime(16,31,27); //2023-01-24T16:31:27
4
5 dataHora = data.atTime(LocalTime.of(16,31,27)); //2023-01-24T16:31:27
6
7 dataHora = data.atTime(LocalTime.of(16,31)); //2023-01-24T16:31
8
9 dataHora = data.atStartOfDay(); //2023-01-24T00:00
```

2.5.7.4 Instant to LocalDateTime

java

```
1 Instant instante = Instant.parse("2023-03-13T15:58:55.794285600Z");
2
3 //Converte instant para LocalDateTime, sem timezone, adicionando zero em offset
4 LocalDateTime dataHora = LocalDateTime.ofInstant(instante, ZoneOffset.UTC);
5 //2023-03-13T15:58:55.794285600
```