# Como manipular datas com o Java 8

## Um dos maiores problemas para os programadores Java sempre foi a manipulação de data, já que a API nativa era limitada. Para resolver esse problema, na versão 8 do Java foi disponibilizada uma nova API com mais funcionalidades.

Em diversas aplicações é necessário manipular datas de alguma forma, seja para salvar a data de algum evento que aconteceu, como a data de nascimento de uma pessoa, a data de admissão de uma pessoa em uma empresa ou de uma festa. Algumas das possíveis operações de datas são: comparação entre elas, criação de um filtro para encontrar um evento que aconteceu entre duas datas e o cálculo do número de dias ou meses que faltam para um evento. Porém, em Java isso sempre foi um problema, pois a API nativa de manipulação de datas sempre foi bastante limitada.

Algumas tentativas para resolver esses problemas surgiram como o [framework JodaTime](https://www.devmedia.com.br/conheca-a-biblioteca-java-joda-time/32110), que tem um conjunto de funcionalidades para a manipulação de datas muito maior que a API do Java até a [versão 7](https://www.devmedia.com.br/java-7-aperfeicoando-o-java-revista-java-magazine-95/22317). Mas apesar de o JodaTime ser bastante útil os programadores [Java](https://www.devmedia.com.br/cursos/java), sempre pediram que a API do Java fosse revista para melhorar o tratamento de datas e isso só aconteceu na última versão do Java, onde toda a API foi revista.

Para mostrar como usar as novas funcionalidades desta API, esse artigo mostra as principais classes, os principais métodos e como utilizar algumas delas. Para a execução dos exemplos desse artigo é necessário ter instalada a [versão 8 da JDK](https://www.devmedia.com.br/java-8-construindo-aplicacoes-web-em-java/30901) e você pode usar qualquer IDE.

### Principais funcionalidades da API

A nova API disponibiliza uma grande quantidade de novas classes e métodos. A classe principal dessa nova especificação é a classe java.time.LocalDate, que tem a mesma função que a antiga classe [java.util.Date](https://www.devmedia.com.br/artigo-java-magazine-22-datas-e-horas-em-java/10125), mas com muito mais funcionalidades implementadas. A **Listagem 1** mostra algumas das possibilidades de utilização da classe LocalDate.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | LocalDate localDate = LocalDate.now();  System.out.println(localDate);  System.out.println("Dia da semana: " + localDate.getDayOfWeek().name());  System.out.println("Dia da semana: " + localDate.getDayOfWeek().ordinal());  System.out.println("Mes: " + localDate.getMonthValue());  System.out.println("Mes: " + localDate.getMonth().name());  System.out.println("Ano: " + localDate.getYear()); |

**Listagem 1**. Classe LocalDate

No código é mostrado como recuperar diversos dados de uma data como o dia da semana, o mês e o ano. Além disso, é possível recuperar o nome do dia da semana, como String (segunda, terça, quarta,...) e também o número (1 a 7). É possível recuperar também o mês e o ano. Tudo isso de uma forma muito mais fácil do que era até na versão 7 do Java e sem nenhuma API adicional. A **Listagem 2** mostra o resultado da execução desse código.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 2016-02-06  Dia da semana: SATURDAY  Dia da semana: 5  Mes: 2  Mes: FEBRUARY  Ano: 2016 |

**Listagem 2**. Execução do código da **Listagem 1**

### Classe Instant

Outra classe nova da API é a classe java.time.Instant, que serve para representar um instante qualquer. Nas versões anteriores do Java, o instante era simplesmente representado por um long, que adicionava um milissegundo a cada instante desde 01/01/1970, que é o instante inicial que a plataforma Java considerava. A classe Instant tem diversos usos, como calcular a duração da execução de algum algoritmo, ou a duração da execução de uma busca, entre diversos outros possíveis usos.

A **Listagem 3** mostra como fazer a comparação de instante na API antiga de datas do Java. O código funciona, mas não é muito elegante, além de só funcionar para eventos a partir de 01/01/1970. O Thread.sleep(1000) foi adicionado apenas para que os dois instantes sejam diferentes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | long instateInicial = System.currentTimeMillis();  try {  Thread.sleep(1000);  } catch (InterruptedException e) {         e.printStackTrace();  }  long instanteFinal = System.currentTimeMillis();  long duracaoEmMilesegundos = instanteFinal - instateInicial;  System.out.println("Duração em segundos: " +  ( duracaoEmMilesegundos / 1000 ) % 60 ); |

**Listagem 3**. Calculando a duração de um evento na API de datas antiga

A **Listagem 4** mostra como fazer essa comparação na nova API com a nova classe Instant. Além de muito mais elegante, o código funciona para qualquer instante recuperado através de uma data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Instant iInicial = Instant.now();  try {         Thread.sleep(1000);  } catch (InterruptedException e) {         e.printStackTrace();  }  Instant iFinal = Instant.now();    Duration duracao = Duration.between(iInicial, iFinal);    System.out.println("Duração em nanos segundos: " + duracao.toNanos());  System.out.println("Duração em minutos: " + duracao.toMinutes());  System.out.println("Duração em horas: " + duracao.toHours());  System.out.println("Duração em milisegundos: " + duracao.toMillis());  System.out.println("Duração em dias: " + duracao.toDays()); |

**Listagem 4**. Calculando a duração de um evento na nova API de datas

Como mostrado no código, é possível recuperar a duração de um evento em nano segundos, minutos, horas e milissegundos, que é outra vantagem da nova API, pois no método antigo o tempo era recuperado sempre em milissegundos e era necessário fazer a conversão dos dados.

A **Listagem 5** mostra o resultado da execução da **Listagem 4** e, como o intervalo é de apenas um segundo, a duração em horas e minutos é arredonda para 0.

|  |  |
| --- | --- |
| 1  2  3  4 | Duração em nanos segundos: 1001000000  Duração em minutos: 0  Duração em horas: 0  Duração em milisegundos: 1001 |

**Listagem 5**. Resultado da execução do código da **Listagem 4**

### Comparação entre datas

Outra coisa bastante utilizada quando trabalhamos com datas é a comparação entre elas como, por exemplo, se uma data é antes ou depois da outra ou quantos meses de diferença existem entre duas datas. Tudo isso é possível fazer com as novas classes da API de data. A **Listagem 6** mostra as principais maneiras de comparar datas com os métodos isAfter, isBefore e isEqual.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | LocalDate localDateAntigo = LocalDate.of(2010, 3, 7);  LocalDate localDateNovo = LocalDate.of(2015, 3,5);    System.out.println(localDateAntigo.isAfter(localDateNovo));  System.out.println(localDateAntigo.isBefore(localDateNovo));  System.out.println(localDateAntigo.isEqual(localDateNovo));    Period periodo = Period.between(localDateAntigo, localDateNovo);  System.out.println(periodo.getYears() + " Anos " + periodo.getMonths() + " Meses " + periodo.getDays() + " Dias");  System.out.println("Apenas meses: " + periodo.toTotalMonths()); |

**Listagem 6**. Comparando duas datas

Veja que a própria classe LocalData tem alguns métodos para a comparação de duas datas, como os métodos isAfter e isBefore, que verificam se uma data é antes ou depois de outra, respectivamente. Para descobrir o tempo passado entre uma data e outra é utilizada a classe Period, onde é possível recuperar a diferença de anos, dias e meses.

A **Listagem 7** mostra o resultado da execução desse trecho de código.

|  |  |
| --- | --- |
| 1  2  3  4 | false  true  4 Anos 11 Meses 26 Dias  Apenas meses: 59 |

**Listagem 7**. Execução do código da **Listagem 6**

Na nova API é também possível fazer operações na Data diretamente na classe LocalDate, não é necessário utilizar o Calendar, como na antiga classe java.util.Date.

A **Listagem 8** mostra o código de algumas possíveis operações na classe LocalDate, dentre elas existem diversas operações como o plusDays(), que adiciona os dias passados como parâmetro a data, o plusWeeks que adiciona as semanas passadas como parâmetro a data e o minusYears(), que remove os anos passados como parâmetro a data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | LocalDate dataManipulacao = LocalDate.now();  System.out.println("Mais 5 dias:" + dataManipulacao.plusDays(5));  System.out.println("Mais 5 semanas:" + dataManipulacao.plusWeeks(5));  System.out.println("Mais 5 anos:" + dataManipulacao.plusYears(5));  System.out.println("Mais 2 meses:" + dataManipulacao.plusMonths(2));  System.out.println("Menos 1 ano:" + dataManipulacao.minusYears(1));  System.out.println("Menos 1 mês:" + dataManipulacao.minusMonths(1));  System.out.println("Menos 3 dia: " + dataManipulacao.minusDays(3));  System.out.println("Data Original:" + dataManipulacao); |

**Listagem 8**. Operações em um LocalDate

Recordando as versões antigas do Java, para fazer essas operações era necessário usar a classe Calendar e os métodos que ela disponibiliza, o que dava bastante trabalho. Na nova API é muito mais simples.

A **Listagem 9** mostra o resultado da execução desse código.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Mais 5 dias: 2016-02-11  Mais 5 semanas: 2016-03-12  Mais 5 anos: 2021-02-06  Mais 2 meses: 2016-04-06  Menos 1 ano: 2015-02-06  Menos 1 mês: 2016-01-06  Data Original: 2016-02-06 |

**Listagem 9**. Execução do código da **Listagem 8**

Uma coisa importante e bem diferente da API antiga e que é possível visualizar no código da **Listagem 8** é que a classe LocalDate funciona como a classe String, isto é, ela é imutável. Quando chamamos um método, como o plusDays(), o objeto dataManipulacao não é alterado, e sim criada uma nova instância da classe LocalDate. É possível verificar isso, pois a última linha do código mostrou a data original e pela execução do código mostrada na **Listagem 9**, é possível perceber que a data não foi modificada.

Outra funcionalidade muito interessante da nova API é utilizar o fuso horário de cidades diferentes para a comparação de datas. Por exemplo, é possível pegar uma data qualquer e definir o fuso horário dela em duas cidades diferentes e depois comparar a diferença de tempo entre essas duas datas considerando o fuso. A **Listagem 10** mostra como fazer isso, onde inicialmente é definida uma hora qualquer, depois é criada a variável horaSaoPaulo utilizando o fuso de São Paulo e a horaParis utilizando o fuso de Paris. Em seguida é feita a comparação entre as duas datas e é mostrada a diferença em horas entre elas.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | LocalDateTime hora = LocalDateTime.of(2016, Month.APRIL, 4, 22, 30);    ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/Sao\_Paulo");  ZonedDateTime horaSaoPaulo = ZonedDateTime.of(hora, fusoHorarioDeSaoPaulo);  System.out.println(horaSaoPaulo);    ZoneId fusoHorarioDeParis = ZoneId.of("Europe/Paris");  ZonedDateTime horaParis = ZonedDateTime.of(hora, fusoHorarioDeParis);  System.out.println(horaParis);    Duration diferencaDeHoras = Duration.between(horaSaoPaulo, horaParis);  System.out.println(diferencaDeHoras.getSeconds() / 60 / 60); |

**Listagem 10**. Comparando datas com fuso horário

A **Listagem 11** mostra a execução do código, onde foram criados dois objetos: um com a hora de São Paulo e outro com a hora de Paris, e depois foi feita a comparação entre essas duas datas. Nessa comparação foi possível ver que a diferença do fuso horário é de cinco horas.

|  |  |
| --- | --- |
| 1  2  3 | 2016-04-04T22:30-03:00[America/Sao\_Paulo]  2016-04-04T22:30+02:00[Europe/Paris]  Diferença de fuso horário: -5 |

**Listagem 11**. Resultado da execução do código da **Listagem 10**

### Formatação de datas

A formatação de data para diferentes padrões também ficou um pouco mais simples nessa nova versão, para isso, agora é criado um formatador de dado com a classe DateTimeFormatter e a própria classe LocalDate tem um método format que retorna uma String com a data formatada no padrão passado como parâmetro. A **Listagem 11** mostra essa classe sendo utilizada e na **Listagem 12** a sua execução.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | LocalDate hoje = LocalDate.now();  DateTimeFormatter formatadorBarra = DateTimeFormatter.ofPattern("dd/MM/yyyy");  DateTimeFormatter formatadorTraco = DateTimeFormatter.ofPattern("dd-MM-yyyy");    System.out.println("Data com /: " + hoje.format(formatadorBarra));  System.out.println("Data com -: " + hoje.format(formatadorTraco)); |

**Listagem 11**. Formatando data com a classe DateTimeFormatter

|  |  |
| --- | --- |
| 1  2 | Data com /: 06/02/2016  Data com -: 06-02-2016 |

**Listagem 12**. Datas formatadas

Repare que foram utilizados dois formatadores: o primeiro que formata a data separando dia, mês e ano com o “/”, e o segundo que formata a data com o -.

Os exemplos anteriores cobriram boa parte da nova API de datas do Java, porém existem muitas outras possibilidades, a API é bastante completa e facilita muito a manipulação de datas. Apenas como exemplo de mais algumas coisas que podem ser feitas, a **Listagem 13** mostra alguns métodos interessantes como, por exemplo, o que verifica se o ano da data é bissexto, o número de dias do mês e do ano, e também a maior e menor data possível na API.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | LocalDate data = LocalDate.now();    System.out.println("Ano bissexto: " + data.isLeapYear());  System.out.println("Número de dias do mês: " + data.lengthOfMonth());  System.out.println("Número de dias do ano: " + data.lengthOfYear());  System.out.println("Menor data: " + LocalDate.MIN);  System.out.println("Maior data: " + LocalDate.MAX); |

**Listagem 13**. Mais alguns métodos interessantes

A **Listagem 14** mostra o resultado da execução do código e como é possível verificar no código, a primeira linha mostra que 2016 é um ano bissexto, a segunda linha mostra que fevereiro tem 29 dias e que o ano de 2016 tem 366 dias. A data máxima suportada pelo Java é +999999999-12-31 e a mínima é -999999999-01-01, em ambas o primeiro número é o ano.

|  |  |
| --- | --- |
| 1  2  3  4  5 | Ano bissexto: true  Número de dias do mês: 29  Número de dias do ano: 366  Menor data: -999999999-01-01  Maior data: +999999999-12-31 |

**Listagem 14**. Execução do código da **Listagem 13**

Para mostrar todo o código desenvolvido nesse exemplo e para mostrar como importar as classes da nova API, a **Listagem 15** mostra o código completo desenvolvido, e que foi mostrado e explicado parte a parte nas listagens anteriores. É possível observar que todas as novas classes incluídas nessa nova API estão contidas no pacote java.time.

**Listagem 15**. Código completo do exemplo desenvolvido nesse artigo.

|  |
| --- |
| package data;    import java.time.Duration;  import java.time.Instant;  import java.time.LocalDate;  import java.time.LocalDateTime;  import java.time.Month;  import java.time.Period;  import java.time.ZoneId;  import java.time.ZonedDateTime;  import java.time.format.DateTimeFormatter;    public class Main {    public static void main(String[] args) {      // criação de datas com a nova classe LocalDate e  LocalDate localDate = LocalDate.now();  System.out.println(localDate);  System.out.println("Dia da semana: " + localDate.getDayOfWeek().name());  System.out.println("Dia da semana: " + localDate.getDayOfWeek().ordinal());  System.out.println("Mes: " + localDate.getMonthValue());  System.out.println("Mes: " + localDate.getMonth().name());  System.out.println("Ano: " + localDate.getYear());    // comparação entre diferentes instantes de tempo  Instant iInicial = Instant.now();  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  Instant iFinal = Instant.now();    Duration duracao = Duration.between(iInicial, iFinal);  System.out.println("Duração em nanos segundos: " + duracao.toNanos());  System.out.println("Duração em minutos: " + duracao.toMinutes());  System.out.println("Duração em horas: " + duracao.toHours());  System.out.println("Duração em milisegundos: " + duracao.toMillis());  System.out.println("Duração em dias: " + duracao.toDays());      // comparação de datas (antes, depois, período entre duas datas)  LocalDate localDateAntigo = LocalDate.of(2010, 3, 7);  LocalDate localDateNovo = LocalDate.of(2015, 3, 5);    System.out.println(localDateAntigo.isAfter(localDateNovo));  System.out.println(localDateAntigo.isBefore(localDateNovo));  System.out.println(localDateAntigo.isEqual(localDateNovo));    Period periodo = Period.between(localDateAntigo, localDateNovo);  System.out.println(periodo.getYears() + " Anos " + periodo.getMonths()  + " Meses " + periodo.getDays() + " Dias");  System.out.println("Apenas meses: " + periodo.toTotalMonths());      // operações em datas como adição e subtração de dias, meses e anos  LocalDate dataManipulacao = LocalDate.now();  System.out.println("Mais 5 dias: " + dataManipulacao.plusDays(5));  System.out.println("Mais 5 semanas: " + dataManipulacao.plusWeeks(5));  System.out.println("Mais 5 anos: " + dataManipulacao.plusYears(5));  System.out.println("Mais 2 meses: " + dataManipulacao.plusMonths(2));  System.out.println("Menos 1 ano: " + dataManipulacao.minusYears(1));  System.out.println("Menos 1 mês: " + dataManipulacao.minusMonths(1));  System.out.println("Menos 3 dia: " + dataManipulacao.minusDays(3));    //classe LocalDate é imutável  System.out.println("Data Original: " + dataManipulacao);    // comparação de datas utilizando o fuso horário  LocalDateTime hora = LocalDateTime.of(2016, Month.APRIL, 4, 22, 30);    ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/Sao\_Paulo");  ZonedDateTime horaSaoPaulo = ZonedDateTime.of(hora, fusoHorarioDeSaoPaulo);  System.out.println(horaSaoPaulo);    ZoneId fusoHorarioDeParis = ZoneId.of("Europe/Paris");  ZonedDateTime horaParis = ZonedDateTime.of(hora, fusoHorarioDeParis);  System.out.println(horaParis);    Duration diferencaDeHoras = Duration.between(horaSaoPaulo, horaParis);  System.out.println("Diferença de fuso horário: " +  diferencaDeHoras.getSeconds() / 60 / 60);    // formatação de datas com a nova API  LocalDate hoje = LocalDate.now();  DateTimeFormatter formatadorBarra = DateTimeFormatter.ofPattern("dd/MM/yyyy");  DateTimeFormatter formatadorTraco = DateTimeFormatter.ofPattern("dd-MM-yyyy");    System.out.println("Data com /: " + hoje.format(formatadorBarra));  System.out.println("Data com -: " + hoje.format(formatadorTraco));      // métodos interessantes  LocalDate data = LocalDate.now();    System.out.println("Ano bissexto: " + data.isLeapYear());  System.out.println("Número de dias do mês: " + data.lengthOfMonth());  System.out.println("Número de dias do ano: " + data.lengthOfYear());  System.out.println("Menor data: " + LocalDate.MIN);  System.out.println("Maior data: " + LocalDate.MAX);      }    } |

Com isso vimos na prática que a nova API de manipulação de datas do Java 8 é bastante poderosa e facilita muito o desenvolvimento de aplicações que precisam considerar datas. Agora o desenvolvedor não precisa gastar tanto tempo com várias classes para fazer o retorno das datas e nem precisa perder tempo com a conversão das mesmas.