**Introdução**

A classe **LocalDate** está localizada no pacote java.time, desde a versão 1.8.

**LocalDate** é um objeto de data imutável (todos os métodos de modificação retornam um novo objeto e, portanto, não alteram o valor do objeto atual) que representa uma data com formato padrão como **yyyy-MM-dd (seguindo ISO 8601)**, fornece também acesso individualmente por exemplo ao dia do ano, dia da semana, semana do ano entre outros.

**Exemplo:** o valor **“1 de Outubro de 2022”** pode ser armazenado em um arquivo do tipo **LocalDate**.

Abaixo segue alguns métodos que serão aplicados em alguns cenários:

**Métodos**

1. **public static LocalDate of()**  
     
   - O método **of()** é usado para obter uma instância de LocalDate a partir do ano, mês e dia do mês passados como parâmetros.   
     
   - O método of() é parecido com o método parse(), mas diferente deste último, que recebe uma data como parâmetro **(parse(yyyy-MM-dd))**, o método of() recebe tipos de parâmetros, conforme a seguir.  
     
   - ano (valor do **tipo inteiro**), mês (que pode ser dois tipos de valores, **int ou Month**, este último é uma classe da API do Java) e ano (valor do **tipo inteiro**).  
   Ex: (**int year, int month, int dayOfMonth) ou  
    (int year, Month month, int dayOfMonth)**  
   - Todos os três parâmetros são obrigatórios

**Retorno:** retorna uma instância de LocalDate de um ano, mês e dia, uma data no tempo local, não nula.

**DateTimeException** - se o resultado exceder a faixa de data suportada

**Sobrecarga: Esse método é sobrecarregado.** (Este método aceita 2 tipos de parâmetros para o mês, int e Month, que representa uma Enum Mês a ser adicionado).

**Exemplos:**

Instanciando uma data usando LocalDate

**LocalDate** data = **LocalDate.of**(2001, 7, 3);

Mostrando o Mês por valor numérico ou por extenso

System.out.println("**LocalDate.of** - [ MES ] -> " + **data.getMonth()**); System.out.println("**LocalDate.of** - [ MES EM VALOR ] -> " + **data.getMonthValue());**

**Demonstração adaptando o uso dos métodos em situações do cotidiano:**

Armazenar datas de um sistema de boletos bancário para poder fazer comparações de vencimento por exemplo.

1. **public static LocalDate plusDays()**   
     
   - O método **plusDays()** retorna uma cópia de LocalDate, com os números de dias adicionados.   
     
   - Este método recebe como parâmetro um valor do tipo **long (plusDays(long daysToAdd).**  
     
   - Este método é similar a outros métodos da classe LocalDate que, incrementa ao valor especificado, determinada quantidade de dias, meses e anos.

**Retorno:** Este método retorna um LocalDate com base nesta data com os dias adicionados, desde que seja não nulo.

**DateTimeException** - se o resultado exceder a faixa de data suportada.

**Sobrecarga:** Esse método **não** é sobrecarregado. (Este método aceita um único parâmetro daysToAdd que representa os dias a serem adicionados, pode ser negativo).

**Exemplos:**

Criando um objeto LocalDate

**LocalDate** data = **LocalDate.parse**("2020-10-02");

Adicionando 5 dias na data

**LocalDate** returnvalue = **data.plusDays**(5);

System.out.println("[Nova data adicionada 5 dias] -> " + returnvalue);

**Demonstração adaptando o uso dos métodos em situações do cotidiano:**

Adicionar dias para fazer alteração em data de vencimento em um sistema de pagamentos por exemplo ou adicionar dias na data atual para um sistema de lembretes.

1. **public static LocalDate minus()**  
     
   - O método **minus()** é usado para obter uma data com o valor especificado subtraído de uma quantidade de unidades, sejam elas em dias meses ou anos.   
     
   - Este método recebe como parâmetro um ou dois tipos, que são:  
   **(TemporalAmount amountToSubtract)** – pode receber um valor do tipo **TemporalAmount**ou **(long amountToSubtract, TemporalUnit unit)** – pode receber dois parâmetros **dos tipos**: **long** e **TemporalUnit**

**Retorno -** Este método retorna o **LocalDate** com base nesta data-tempo com a quantidade especificada de unidades subtraída, em dias, semanas, meses e anos.

**DateTimeException** - Se não for possível subtrair o valor, pois a unidade não é suportada ou por algum outro motivo, uma exceção é lançada.

**Sobrecarga:** Esse método é sobrecarregado, uma vez que aceita tipos e quantidade de parâmetros diferentes.

**Exemplos:**

Criando um objeto LocalDate

**LocalDate** data **= LocalDate.parse**("2020-10-02");

Diminuindo 12 Anos na data

**LocalDate** returnvalue = **data.minus**(12, ChronoUnit.YEARS);

System.out.println("[Nova data diminuída 12 anos] -> " + returnvalue);

**Demonstração adaptando o uso dos métodos em situações do cotidiano:**

Subtraindo datas para fazer alteração em data de vencimento em um sistema de pagamentos por exemplo ou diminuir data para um sistema de lembrete.