**Introdução**

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

**LocalDate** é um objeto de data e hora 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 **aaaa-MM-dd**, fornece também acesso individualmente por exemplo ao dia do ano, dia da semana e semana do ano. **Exemplo:** o valor “1 de Outubro de 2022” pode ser armazenado em um arquivo **LocalDate**.

Abaixo segue alguns métodos que serão aplicados em cenários como:

**Métodos**

1)

**Public static LocalDate of(int year, int month, int dayOfMonth) ou Public static LocalDate of(int year, Month month, int dayOfMonth)** -> O método of(int, Month, int) da classe LocalDate em Java é usado para obter uma instância de LocalDate a partir do ano, mês e dia do mês de entrada. Neste método, todos os três parâmetros são obrigatórios e são passados ​​na forma de inteiro o ano, mês e dia do mês ou inteiro ano, Month mês e inteiro dia.

**Retorno:** obtém 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.

**2)**

**Public static LocalDate plusDays(long daysToAdd)**-> O método plusDays(long daysToAdd) da classe LocalDate em Java é usado para obter uma cópia com o número especificado de dias adicionados. Este método adiciona o valor especificado ao campo de dias incrementando os campos de mês e ano conforme necessário para garantir que o resultado permaneça válido.

**Retorno:** Este método retorna um LocalDate com base nesta data com os dias adicionados, 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 lembrete.

**3)**

**Public static LocalDate minus(TemporalAmount amountToSubtract) ou minus(long amountToSubtract, TemporalUnit unit)->** O método minus(long daysToAdd) da classe LocalDate em Java é usado para obter uma desta data com o valor especificado subtraído. Este método adiciona o valor especificado ao campo de dias incrementando os campos de mês e ano conforme necessário para garantir que o resultado permaneça válido.

**Retorno-** Este método retorna o **LocalDate** com base nesta data-tempo com a quantidade especificada subtraída.

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

**Sobrecarga:** Esse método é sobrecarregado. (Este método aceita tipos e quantidade de parâmetros diferentes como o mesmo nome de método).

**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.