Datas e Controle de Fluxo

Nivelamento de Lógica de Programação e Programação Orientada a Objeto

Java possui duas API's para datas, sendo uma legada (*Date*) e a mais nova introduzida no JDK 8 (*java.time*). Vamos começar pela forma atual e depois a legada.

A nova API segue a padronização da **ISO 8601**.

Podemos dividir esse estudo em três partes:

- **human-friendly:** é a que conseguimos ler e tem fácil manipulação.
- **computacional:** tem o foco na precisão.
- **cálculos:** são relacionados às comparações, etc.

- **LocalTime:** representa uma hora do dia com os valores para hora, minuto, segundo e milissegundos.
- **LocalDate:** representa uma data com dia, mês e ano.
- **LocalDateTime:** representa uma data e hora.
- **Month/Year/DayOfWeek/YearMonth:** auxiliares de LocalDate para manipulações mais específicas.
- **Instant:** usado para cálculos computacionais com precisão de nanossegundos.
- **Duration:** diferença de tempo e cálculos baseados em horas.
- **Period:** diferença de datas e cálculos baseados em datas.

Ainda existem muitas outras classes como a **OffsetDateTime** e **ZonedOffset** para representação de fusos-horários (timezones) e o pacote *java.time.temporal* para acessar unidades de tempo e cálculos, como a **ChronoUnit** que permite calcular a diferença de tempo de acordo com uma unidade específica, como a diferença entre duas datas em dias, anos, minutos, etc.

Instâncias

Controlando o Fluxo de Execução

Instância

As classes do pacote java.time são imutáveis, então não é possível instanciar e modificar seus atributos.

Assim, o que temos são características dos padrões de projeto *Singleton* e *Builder*, onde acessamos uma instância ou podemos definir como é o objeto que queremos manipular.

Parece a mesma coisa, mas não temos nunca acesso ao construtor das classes.

```
import java.time.LocalDate;

public class Application {
   public static void main(String[] args) {
       LocalDate hoje = LocalDate.now();//Obtém a representação da data de hoje
       LocalDate independencia = LocalDate.of(1822, 9, 7);//Obtém uma data específica
   }
}
```

Conversão "de" e "para" texto

Controlando o Fluxo de Execução

Conversão "de" e "para" texto

Outra forma possível de obter um `LocalDate` é convertendo a partir de texto. Para isso é necessário definir o formato da data.

No trecho abaixo, estão dois formatos.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Application {
   public static void main(String[] args) {
       LocalDate olimpiadas = LocalDate.parse("26/07/2024", DateTimeFormatter.ofPattern("dd/MM/yyyy"));
       LocalDate copaFutebol = LocalDate.parse("2026-06-11", DateTimeFormatter.ISO_DATE);
   }
}
```

Controlando o Fluxo de Execução

Conversão "de" e "para" texto

Da mesma forma, usamos os formatadores para converter de `LocalDate` para texto:

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Application {
    public static void main(String[] args) {
        LocalDate olimpiadas = LocalDate.parse("26/07/2024", DateTimeFormatter.ofPattern("dd/MM/yyyy"));
        LocalDate copaFutebol = LocalDate.parse("2026-06-11", DateTimeFormatter.ISO_DATE);

        String sOlimpiadas = olimpiadas.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"));
    }
}
```

Marcadores de formatação

A seguir, as marcações mais comuns no uso do `DateTimeFormatter`:

- **d:** dia com 1 dígito (1, 2, 10, 30)
- **dd:** dia com 2 dígitos (01, 02, 10, 30)
- **MM:** mês com 2 dígitos (01, 02, 10, 12)
- **MMM:** abreviação do nome do mês (Jan, Feb, Dec)
- **MMMM:** nome completo do mês (January, February, December)
- **yy:** ano com 2 dígitos (99, 23)
- **yyyy:** ano com 4 dígitos (1999, 2023)
- **H:** hora do dia (0-23) com 1 dígito (0, 1, 14, 23)
- **HH:** hora do dia (0-23) com 2 dígitos (00, 01, 14, 23)
- **h:** hora em formato de 12 horas (1-12) com 1 dígito (1, 2, 10, 12)
- **hh:** hora em formato de 12 horas (1-12) com 2 dígitos (01, 02, 10, 12)
- **m:** minutos com 1 dígito (0, 5, 30)
- **mm:** minutos com 2 dígitos (00, 05, 30)
- **s:** segundos com 1 dígito (0, 5, 45)

A seguir, as marcações mais comuns no uso do `DateTimeFormatter`:

- **ss:** segundos com 2 dígitos (00, 05, 45)
- **S:** fração de segundo com 1 dígito (3 para 0.3 segundos)
- **SSS:** fração de segundo com 3 dígitos (003 para 0.003 segundos)
- **a:** marca AM/PM (AM, PM)
- **z:** fuso horário abreviado (PST, CET)
- **Z:** fuso horário numérico com sinal (+0000, -0800)
- **X:** fuso horário numérico com sinal e dois dígitos de hora (Z, +08, -04)
- **XX:** fuso horário numérico com sinal, dois dígitos de hora e dois dígitos de minutos (Z, +0800, -0430)
- **XXX:** fuso horário numérico com sinal, dois dígitos de hora e dois dígitos de minutos separados por ":" (Z, +08:00, -04:30)
- **E:** abreviação do dia da semana (Mon, Tue, Fri)
- **EEEE:** nome completo do dia da semana (Monday, Tuesday, Friday)

Internacionalização

Controlando o Fluxo de Execução

Internacionalização

Na sessão anterior você pode ter percebido que os dias da semana e meses estão em inglês.

Esse é o padrão da linguagem, mas não significa que você precisa traduzir ou fazer algum código de conversão.

A API `java.time` trabalha muito bem com internacionalização usando `java.util.Locale`.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class Application {
    public static void main(String[] args) {
        LocalDate hoje = LocalDate.now();
        String diaDaSemana = hoje.format(DateTimeFormatter.ofPattern("EEEE", Locale.of("pt", "BR")));
        System.out.println(diaDaSemana);
    }
}
```

Date (legado)

Controlando o Fluxo de Execução

Date (legado)

As primeiras versões do Java contavam com a class java.util.Date para manipulação de datas. Seu funcionamento era baseado na *interface Calendar* e não possuía uma representação uniforme, além de não trabalhar bem com padrões de internacionalização.

Alguns problemas conhecidos são o mês de janeiro representado como mês 0 (zero) e os anos são representados a partir de 1900, ou seja, 2025 seria 125. Outro ponto é que a classe *Date* não é imutável, podendo causar problemas na manipulação e no comportamento.

Ainda assim, pode ser possível precisar manter códigos que usam essa classe. Se a aplicação estiver numa versão superior ao Java 8, você pode converter das seguintes formas

```java
import java.util.Date;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;

public class Converter {

    // Converte java.util.Date para LocalDate
    public static LocalDate dateToLocalDate(Date date) {
        return date.toInstant()
                .atZone(ZoneId.systemDefault())
                .toLocalDate();
    }

    // Converte java.util.Date para LocalDateTime
    public static LocalDateTime dateToLocalDateTime(Date date) {
        return date.toInstant()
                .atZone(ZoneId.systemDefault())
                .toLocalDateTime();
    }

    // Converte LocalDate para java.util.Date
    public static Date localDateToDate(LocalDate localDate) {
        return Date.from(localDate.atStartOfDay()
                .atZone(ZoneId.systemDefault())
                .toInstant());
    }

    // Converte LocalDateTime para java.util.Date
    public static Date localDateTimeToDate(LocalDateTime localDateTime) {
        return Date.from(localDateTime.atZone(ZoneId.systemDefault())
                .toInstant());
    }
}
```

Controle de fluxo - Condicionais

Controlando o Fluxo de Execução

Em Java, temos blocos especiais que controlam o fluxo de execução do nosso programa. Ou seja, esses blocos podem ser executados sim/não de acordo com uma condição definida (`if-then`, `if-then-else`, `switch`) ou ainda, podem ser executados repetidamente (`for`, `while`, `do-while`) de acordo com o controle definido.

Estrutura Condicionais

Controlando o Fluxo de Execução

A estrutura mais simples é o bloco if-then. O conteúdo deste bloco será executado apenas quando a condição definida for true. No seguinte trecho de código, observamos uma representação de uma funcionalidade de frenagem de uma bicicleta. Assim, se o valor de isMoving for verdadeiro, então a velocidade currentSpeed será decrementada.

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

Estrutura Condicionais

Controlando o Fluxo de Execução

Fazendo a alteração no código anterior, temos

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

Estrutura Condicionais

Controlando o Fluxo de Execução

Estruturas `if-then-else` podem ser encadeadas usando a combinação de palavras-chave `else if`. Veja no próximo exemplo o algoritmo para converter uma nota de 0-100 em graduações A-F. A saída esperada é `Grade = C`.

Estrutura Condicionais

Controlando o Fluxo de Execução

```
class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 76;
        char grade;
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

Estrutura Condicionais

Controlando o Fluxo de Execução

Existe um cenário especial, semelhante ao código anterior, quando desejamos realizar determinado comportamento de acordo com o valor contido em uma variável, sendo esse valor pertencente a um universo bem limitado. Por exemplo, desejamos converter valores 1-6 em texto com o respectivo nome do mês.

Estrutura Condicionais

Controlando o Fluxo de Execução

```java
public class SwitchDemo {
    public static void main(String[] args) {
        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
}
```

Enums

O Java possui um tipo especial que são as *enumerations*, ou apenas **enums**. Esse tipo representa valores finitos de uma determinada representação. Por exemplo, dias da semana.

Podemos escrever um *enum* contendo os sete dias da semana, da seguinte forma:

```
public enum DiaDaSemana {

    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA

}
```

Exercícios

Exercício 1: Emenda de Feriado

Dado um feriado, se tiver emenda (terça ou quinta-feira), vamos planejar uma viagem.

Desafio

Calcular quantos dias, dias úteis (seg a sex) e semanas faltam de hoje até a data da viagem.

Exercício 2: Calculando Idade

Desenvolva um programa que receba uma data de nascimento e calcule a idade da pessoa.

Mostre o resultado em anos, meses e dias.

Exercício 3: Datas Futuras

Crie um programa que defina a data de hoje, adicione 15 dias a ela e mostre o resultado formatado como "dd/MM/yyyy".

Exercício 4: Data e Hora Formatadas

Crie um `LocalDateTime` para o momento atual.

Formate e imprima a data e hora no seguinte formato:

`"dd 'de' MMMM 'de' yyyy 'às' HH:mm:ss"`

Exercício 5: Enum com Switch-Case e Scanner

Crie um enum chamado `OperacaoMatematica` representando operações matemáticas básicas (+, -, *, /).

Utilize um Scanner para obter a operação do usuário e realizar o cálculo com dois números.

Exercício 6: Enum com Switch-Case e Scanner

Crie um enum chamado **CategoriaProduto** representando categorias de produtos (ELETRONICO, VESTUARIO, ALIMENTO).

Utilize um Scanner para obter a categoria do usuário e imprima uma mensagem correspondente.

Exercício 7: Enum com Switch-Case e Scanner

Crie um enum chamado **UnidadeTemperatura** representando unidades de temperatura (CELSIUS, FAHRENHEIT, KELVIN).

Escreva um programa que utiliza um Scanner para obter a unidade de temperatura atual do usuário e a unidade desejada, pedindo a temperatura atual e imprimindo a temperatura convertida.

CELSIUS:

F → (*celsius* * 9 / 5) + 32

K → *celsius* + 273.15

KELVIN:

C → *kelvin* - 273.15

F → (*kelvin* * 9 / 5) - 459.67

FAHRENHEIT:

C → (*fehrenheit* - 32) * 5 / 9

k → (*fehrenheit* + 459.67) * 5 / 9

Obrigada