SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

Java - OO

Desenvolvimento de Sistemas

Prof. Me. Reneilson Santos

Março/2024

SENAI
PELO FUTURO DO TRABALHO

Agenda

- ➔ Enumeradores
- ➔ Data
- ➔ Interfaces
- ➔ Relacionamentos

Enumeradores

Enumeradores

São tipos de campos que consistem em **um conjunto fixo de constantes** (static final), sendo como **uma lista de valores pré-definidos**.

Em Java, um enumerador pode ser definido como um tipo de enumeração usando a palavra chave **enum**.

Todos os tipos enums implicitamente estendem a classe java.lang.Enum.

Enum

```
public enum MarcaCarro {
 FERRARI,
 CHEVROLET,
 FIAT,
 FORD,
 TESLA,
 PORSCHE,
 HONDA,
 BMW,
 NISSAN,
 VOLVO,
 KIA,
 SUZUKI,
 MITSUBISHI,
 LANDROVER,
 PEUGEOT,
 HYUNDAI,
 CITROEN,
}
```

```
public enum Cor {
 PRETO,
 VERMELHO,
 AZUL,
 PRATA,
 AMARELO,
 VERDE,
 VINHO,
 BRANCO,
}
```

Enum

```
public class Carro {
 private MarcaCarro marca;
 private Cor cor;
 public Carro(MarcaCarro marca, Cor cor) {
   this.marca = marca;
   this.cor = cor;
 }
 public MarcaCarro getMarca() {
   return marca;
 }
 public void setMarca(MarcaCarro marca) {
   this.marca = marca;
 }
 public Cor getCor() {
   return cor;
 }
 public void setCor(Cor cor) {
   this.cor = cor;
 }
}
```

Enum

```
public static void main(String[] args){
    Carro carro = new Carro(MarcaCarro.BMW, Cor.VERMELHO);
    System.out.println(carro.toString());
    // Carro [marca=BMW, cor=VERMELHO]
}
```

Data

Data

O Java 8 trouxe consigo uma reformulação significativa no tratamento de datas, introduzindo o pacote **java.time**.

Este pacote apresenta diversas classes que oferecem uma ampla variedade de recursos para o manuseio de datas e tempos.

Entre as classes mais importantes estão **LocalDate**, **LocalTime**, **LocalDateTime**, **ZonedDateTime**, **Duration**, entre outras.

```java
import java.time.LocalDate;

public class DataJava {
    public static void main(String[] args) {
        // Obtendo a data atual
        LocalDate hoje = LocalDate.now();
        System.out.println("Data Atual: " + hoje);

        // Adicionando 7 dias à data atual
        LocalDate daquiASeteDias = hoje.plusDays(7);
        System.out.println("Daqui a 7 dias: " + daquiASeteDias);

        // Verificando se uma data está no passado
        LocalDate dataPassada = LocalDate.of(2022, 1, 1);
        System.out.println("Está no passado? " + hoje.isAfter(dataPassada));
    }
```

Formatando Data em Java

A biblioteca **java.time** introduzida no Java 8 trouxe um novo conjunto de classes para manipulação de datas e horas, e **DateTimeFormatter** é particularmente útil para formatar e analisar datas em diferentes representações de string.

Letra	Componente de Data ou Tempo	Tipo	Exemplos
G	Era	Texto	AD
y	Ano	Ano	1996; 96
M	Mês do ano	Mês	July; Jul; 07
w	Semana do ano	Número	27
W	Semana do mês	Número	2
D	Dia do ano	Número	189
d	Dia do mês	Número	10
F	Dia da semana do mês	Número	2
E	Dia da semana	Texto	Tuesday; Tue
a	Marcador AM/PM	Texto	PM
H	Hora do dia(0-23)	Número	0
k	Hora do dia(1-24)	Número	24
K	Hora em am/pm(0-11)	Número	0
h	Hora em am/pm(1-12)	Número	12
m	Minutos de hora	Número	30
s	Segundos de minutos	Número	55
S	Millisegundo	Número	978
z	Time zone	General Time Zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

```java
public class TransformarDataEmString {
    public static void main(String[] args) {
        // Criar uma instância de LocalDate
        LocalDate data = LocalDate.now();

        // Definir um formato desejado
        DateTimeFormatter formatoData = DateTimeFormatter.ofPattern( "dd/MM/yyyy");

        // Transformar a data em uma string usando o formato especificado
        String dataFormatada = data.format(formatoData);

        // Imprimir a data formatada
        System.out.println( "Data formatada: " + dataFormatada);
    }
}
```

Interfaces

Interfaces

Uma interface, em termos simples, é um **contrato que define** um conjunto de métodos que uma classe deve implementar.

Ela atua como um ponto de encontro entre diferentes componentes de um sistema, garantindo que certos comportamentos sejam obedecidos, promovendo uma abordagem consistente e modular na construção de software.

Ao empregar interfaces, ganhamos a habilidade de separar a declaração do comportamento real. Isso significa que podemos criar classes independentes que implementam uma interface específica, promovendo um desacoplamento eficaz.

Exemplo

```
public interface ICalculadora {
    float soma(float a, float b);
    float subtracao(float a, float b);
    float divisao(float a, float b);
    float multiplicacao(float a, float b);
    float resto(float a, float b);
}
```

Uso de Interface

```java
public class Calculadora implements ICalculadora{
    @Override
    public float somar(float a, float b) {
        return a + b;
    }
    @Override
    public float subtrair(float a, float b) {
        return a - b;
    }
    @Override
    public float multiplicar(float a, float b) {
        return a * b;
    }
    @Override
    public float dividir(float a, float b) {
        return a / b;
    }
    @Override
    public float resto(float a, float b) {
        return a % b;
    }
}
```

Relacionamentos

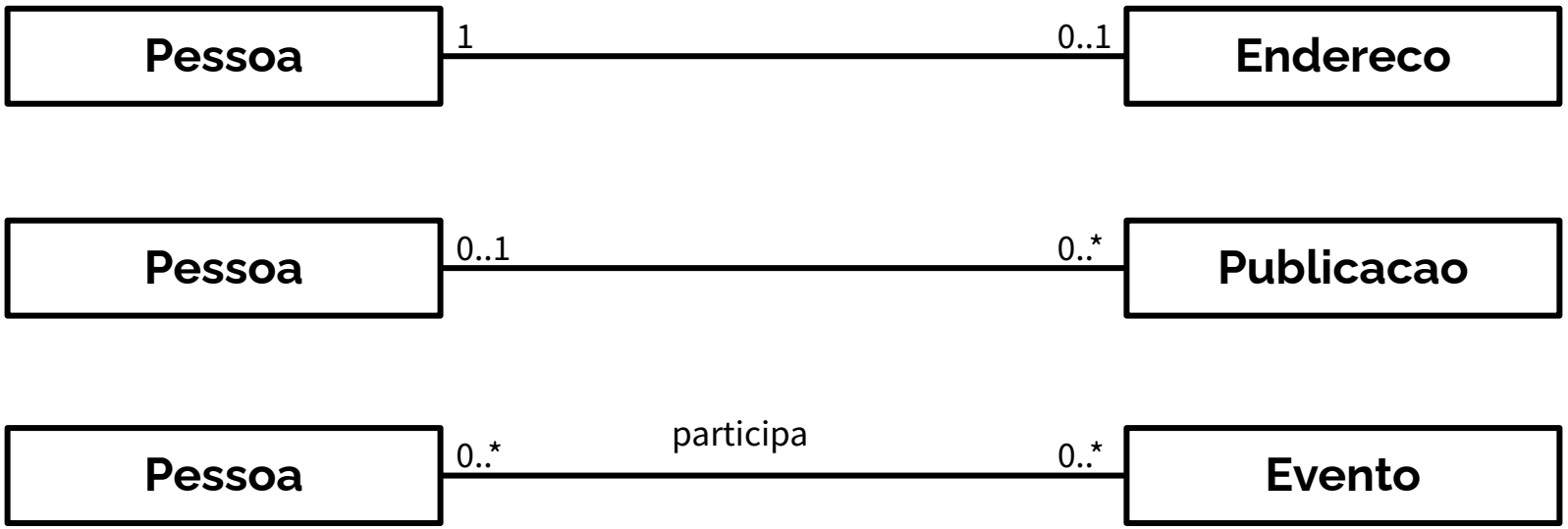
Relacionamentos

Assim como nos bancos de dados, relacionamentos entre classes em programação orientada a objetos definem a interação entre diferentes partes do sistema.

Estabelecem vínculos entre objetos, promovendo coesão e reutilização de código.

Esses relacionamentos, como associações e heranças, são cruciais para estruturar sistemas eficientes e modularizados.

Relacionamentos



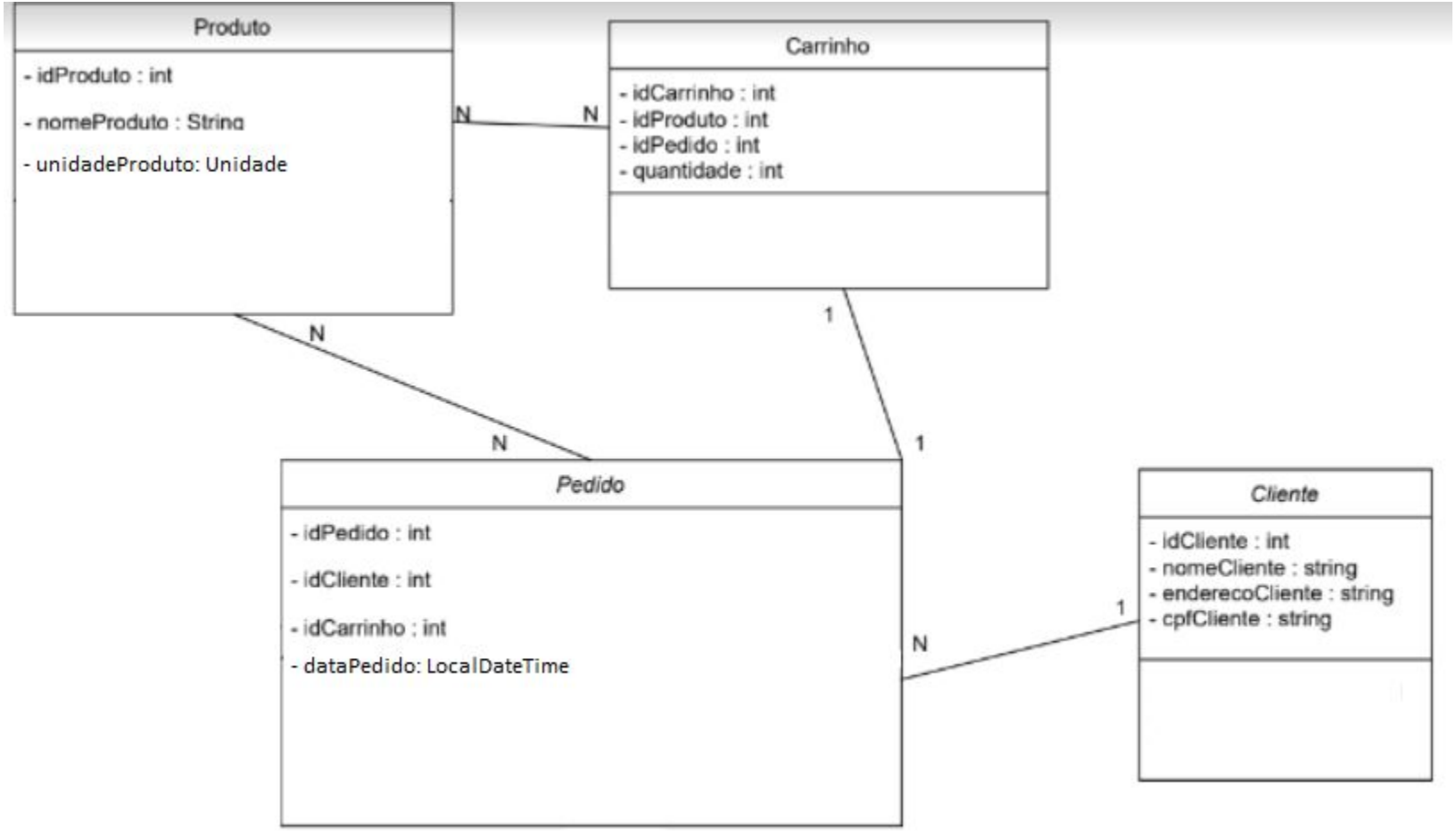
Exemplo : Todos Relacionamentos Mapeados em Pessoa

```
public class Pessoa {
    private Endereco endereco;
    private List<Publicacao> publicacoes;
    private List<Evento> eventos;
}
```

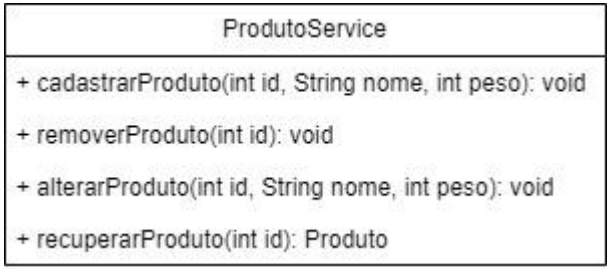
Atividade 1

Desenvolver o seguinte diagrama de classes.

O Enumerador Unidade tem os seguintes valores: KG, M, L. (Quilograma, metros e litros)


Produto
- idProduto : int
- nomeProduto : String
- unidadeProduto: Unidade
Carrinho
- idCarrinho : int
- idProduto : int
- idPedido : int
- quantidade : int
N
N
N
1
N
1
Pedido
- idPedido : int
- idCliente : int
- idCarrinho : int
- dataPedido: LocalDateTime
N
1
Cliente
- idCliente : int
- nomeCliente : string
- enderecoCliente : string
- cpfCliente : string

Atividade 2



Criar uma interface para o serviço de produto indicado no diagrama.