

Curso: Modelagem Conceitual com Diagrama de Classes da UML

<https://www.udemy.com/user/nelio-alves>

Prof. Dr. Nelio Alves

Capítulo: Estudo de caso: implementação Java com Spring Boot e JPA

Notas de aula do estudo de caso

Requisitos:

Necessário:

- Computador Mac, Linux ou Windows
- Conhecimento básico de operação do sistema operacional: instalação / descompactação / terminal
- Conhecimento básico de Programação Orientada a Objetos em alguma linguagem moderna (Java, C#, Python, PHP, etc.): classes, atributos, métodos, construtores, encapsulamento, elementos estáticos.

Desejável:

- Conhecimento básico de Git

Objetivo geral:

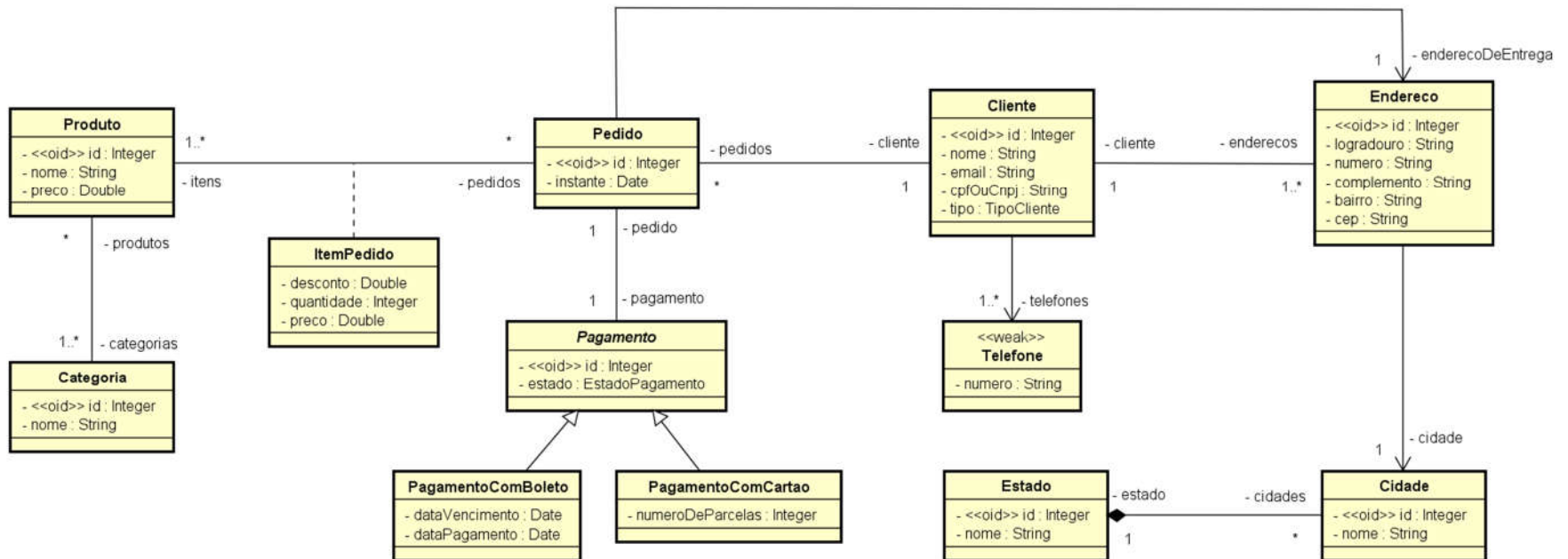
Este estudo de caso tem como objetivo mostrar na prática como um modelo conceitual pode ser implementado sobre o paradigma orientado a objetos, usando padrões de mercado e boas práticas.

Vamos tomar como caso um modelo conceitual abrangente, com o qual possamos mostrar a implementação prática em linguagem orientada a objetos dos tópicos aprendidos no curso, quais sejam:

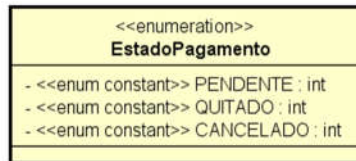
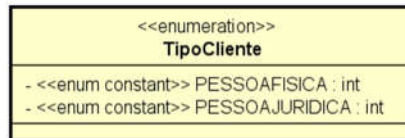
- Leitura e entendimento do diagrama de classes
- Leitura e entendimento do diagrama de objetos
- Associações
- Um para muitos / muitos para um
- Um para um
- Muitos para muitos
- Conceito dependente
- Classe de associação
- Herança
- Enumerações
- Tipos primitivos (ItemPedidoPK)
- Entidades fracas (ElementCollection)
- Associações direcionadas

Objetivos específicos:

1) Fazer uma implementação padrão do seguinte modelo conceitual:

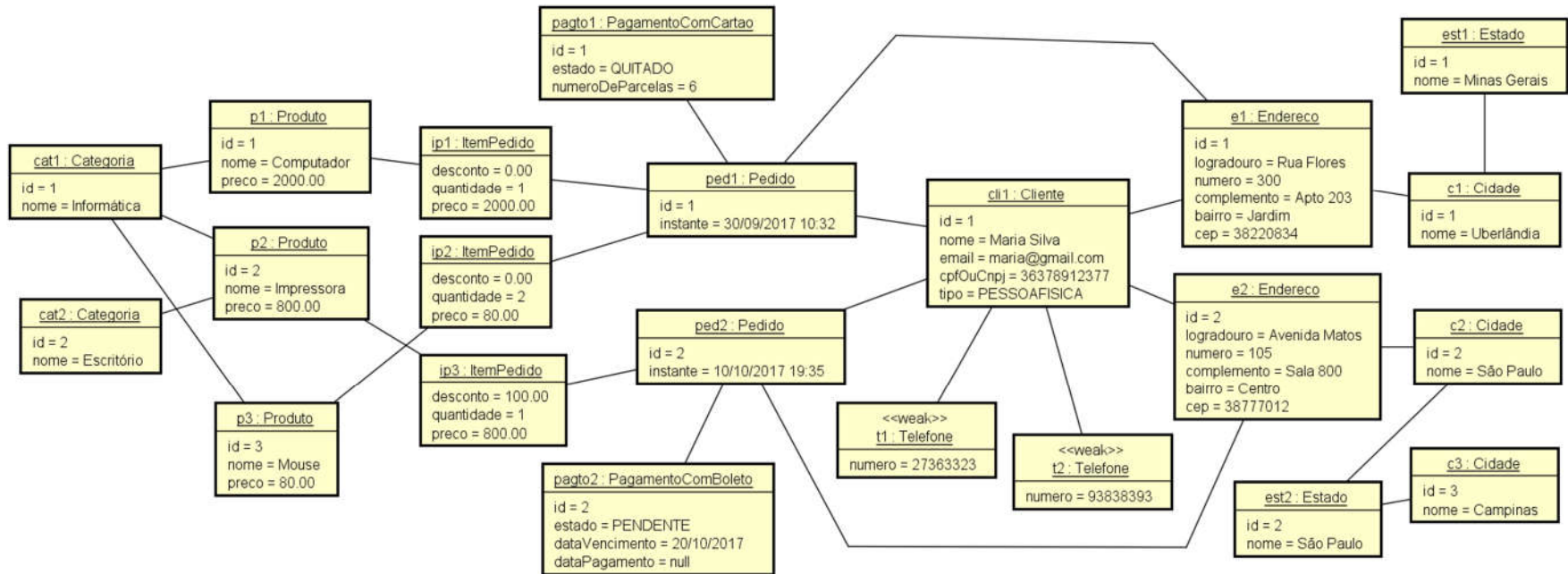


Enumerações:



Objetivos (continuação):

2) Criar a seguinte instância do modelo conceitual:



3) Gerar uma base de dados relacional automaticamente a partir do modelo conceitual, bem como povoar a base com a instância dada.

4) Recuperar os dados e disponibilizá-los por meio de uma API Rest BÁSICA. Os seguintes *end points* devem ser disponibilizados:

End point	Dados
/categorias/{id}	Categoria e seus produtos
/clientes/{id}	Cliente, seus telefones e seus endereços
/pedidos/{id}	Pedido, seu cliente, seu pagamento, seus itens de pedido, seu endereço de entrega

Instalando e testando as ferramentas

- Git
- Conta no Github
- Google Chrome e Postman
- JDK - Java Development Kit
- STS - Spring Tool Suit (Eclipse / Maven / Tomcat / Jackson / JPA)

Ajuste do layout do STS:

- Window -> Perspective -> Open Perspective -> Other -> Spring
- Window -> Perspective -> Reset Perspective
- Minimizar as abas Outline e Spring Explorer

Criando e testando o projeto

- Botão direito na área da aba Package Explorer -> New -> Spring Starter Project
 - Se não aparecer: New -> Other -> Procure
- Opções:
 - Name: cursomc
 - Type: Maven
 - Java Version: 1.8
 - Group: com.nelioalves.cursomc
 - Artifact: cursomc
 - Version: 1.0.0-SNAPSHOT (é uma convenção do Maven)
 - Description: Estudo de caso Java para curso de Modelagem Conceitual com UML
 - Package: com.nelioalves.cursomc
 - Next
 - Web -> Web
- Botão direito -> Run As -> Spring Boot App
SE OCORRER UM ERRO PORQUE A PORTA 8080 JÁ ESTÁ EM USO, OU PARE A APLICAÇÃO,
OU MUDE A PORTA:
application.properties:
server.port=\${port:8081}

Primeiro commit: Projeto criado

- **Iniciar um repositório de versionamento na pasta do projeto:**
`git init`
- **Configurar usuário e email (use seu email do Github):**
`git config --global user.name "Seu nome"`
`git config --global user.email "seuemail@seudominio"`
- **Fazer o primeiro commit:**
`git add .`
`git commit -m "Projeto criado"`

Commit: Testando o REST

- **Arrumando o problema do atalho CTRL + SHIFT + O:**
 - Preferences -> General -> Keys
 - Filters -> desmarque Filter uncategorized commands
 - Localize "Go To Symbol in File", selecione-o e clique "unbind"
 - Apply / Close
- **Classe CategoriaResource (subpacote resources)**

```
package com.nelioalves.cursomc.resources;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/categorias")
public class CategoriaResource {

    @RequestMapping(method=RequestMethod.GET)
    public String listar() {
        return "REST está funcionando!";
    }
}
```

Commit: Testando a primeira classe de dominio - Categoria

- **Checklist para criar entidades:**
 - Atributos básicos
 - Associações (inicie as coleções)
 - Construtores (não inclua coleções no construtor com parâmetros)
 - Getters e setters
 - hashCode e equals (implementação padrão: somente id)
 - Serializable (padrão: 1L)
- **Método listar atualizado:**

```
@RequestMapping(method=RequestMethod.GET)
public List<Categoria> listar() {
    Categoria cat1 = new Categoria(1, "Informática");
    Categoria cat2 = new Categoria(2, "Escritório");

    List<Categoria> lista = new ArrayList<>();
    lista.add(cat1);
    lista.add(cat2);

    return lista;
}
```

Commit: Banco de dados H2 e criação automática da base de dados

- **Dependências:**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

- **Rodar /h2-console com a base `jdbc:h2:mem:testdb`**
- **Mapeamento da classe Categoria:**

```
@Entity
public class Categoria implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
```

- **Alterar o arquivo application.properties**

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

ATUALIZAÇÃO - H2 em algumas versões novas:

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.datasource.url=jdbc:h2:file:~/test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
```

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# No JDBC URL: jdbc:h2:file:~/test
```

Commit: Criando repository e service para Categoria

(vide repositório e conteúdo do vídeo)

Commit: Criando operacao de instanciacao

(vide repositório e conteúdo do vídeo)

Commit: Produto e associacao muitos para muitos

- **Mapeamento na classe Produto:**

```
@ManyToMany
@JoinTable(name = "PRODUTO_CATEGORIA",
    joinColumns = @JoinColumn(name = "produto_id"),
    inverseJoinColumns = @JoinColumn(name = "categoria_id")
)
private List<Categoria> categorias = new ArrayList<>();
```

- **Mapeamento na classe Categoria:**

```
@ManyToMany(mappedBy = "categorias")
private List<Produto> produtos = new ArrayList<>();
```

Commit: Ajustes no endpoint /categorias/{id}

- **Proteção para referência cíclica na serialização Json:**

```
@JsonManagedReference
@JsonBackReference
```

- **Checklist de tratamento de exceção de id inválido:**

- Criar ObjectNotFoundException
- Criar StandardError
- Criar ResourceExceptionHandler

Commit: Estado e Cidade

- **Checklist para criar entidades:**
 - Atributos básicos
 - Associações (inicie as coleções)

- Construtores (não inclua coleções no construtor com parâmetros)
- Getters e setters
- hashCode e equals (implementação padrão: somente id)
- Serializable (padrão: 1L)

- **Mapeamentos:**

```
@Entity
public class Cidade implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nome;

    @ManyToOne
    @JoinColumn(name="estado_id")
    private Estado estado;
```

```
@Entity
public class Estado implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nome;

    @OneToMany(mappedBy="estado")
    private List<Cidade> cidades;
```

Commit: Cliente, TipoCliente, telefones e enderecos

- **Implementação do Enum:**

```
package com.nelioalves.cursomc.domain.enums;

public enum TipoCliente {

    PESSOA FISICA(1, "Pessoa Física"),
    PESSOA JURIDICA(2, "Pessoa Jurídica");

    private int cod;
    private String descricao;

    private TipoCliente(int cod, String descricao) {
        this.cod = cod;
        this.descricao = descricao;
    }

    public int getCod() {
        return cod;
    }
```



```

    }

    public String getDescricao() {
        return descricao;
    }

    public static TipoCliente toEnum(Integer id) {

        if (id == null) {
            return null;
        }

        for (TipoCliente x : TipoCliente.values()) {
            if (id.equals(x.getCod())) {
                return x;
            }
        }
        throw new IllegalArgumentException("Id inválido " + id);
    }
}

```

- Definição do tipo do cliente e seu getter e setter:

```

private Integer tipo;

public TipoCliente getTipo() {
    return TipoCliente.toEnum(tipo);
}

public void setTipo(TipoCliente tipo) {
    this.tipo = tipo.getCod();
}

```

- Mapeamento dos telefones (ElementCollection):

```

@ElementCollection
@CollectionTable(name = "TELEFONE")
private Set<String> telefones = new HashSet<>();

```

Commit: Endpoint /clientes/{id} disponível

- Checklist:
 - Criar ClienteService
 - Criar ClienteResource
 - Proteger contra serialização Json cíclica

Commit: Pedido, EstadoPagamento e Pagamento

- **Nota: Mapeamentos de herança:**
<https://www.thoughts-on-java.org/complete-guide-inheritance-strategies-jpa-hibernate/>

- **Classe Pedido:**

```
@Entity
public class Pedido implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @Temporal(TemporalType.TIMESTAMP)
    private Date instante;

    @OneToOne(cascade = CascadeType.ALL, mappedBy="pedido")
    private Pagamento pagamento;

    @ManyToOne
    @JoinColumn(name="cliente_id")
    private Cliente cliente;

    @ManyToOne
    @JoinColumn(name="endereco_id")
    private Endereco enderecoDeEntrega;
```

- **Classe Pagamento:**

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Pagamento implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private Integer id;

    private Integer estado;

    @JoinColumn(name="pedido_id")
    @OneToOne
    @MapsId
    private Pedido pedido;
```

- **Classe PagamentoComBoleto:**

```
@Entity
public class PagamentoComBoleto extends Pagamento {
    private static final long serialVersionUID = 1L;

    @Temporal(TemporalType.DATE)
    private Date dataVencimento;

    @Temporal(TemporalType.DATE)
    private Date dataPagamento;
```

```

    public PagamentoComBoleto() {
    }

```

- **Classe PagamentoComCartao:**

```

@Entity
public class PagamentoComCartao extends Pagamento {
    private static final long serialVersionUID = 1L;

    private Integer numeroDeParcelas;

```

- **Instanciação:**

```

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy hh:mm");

Pedido ped1 = new Pedido(null, sdf.parse("30/09/2017 10:32"), cli1, e1);
Pedido ped2 = new Pedido(null, sdf.parse("10/10/2017 19:35"), cli1, e2);

cli1.getPedidos().addAll(Arrays.asList(ped1, ped2));

Pagamento pagto1 = new PagamentoComCartao(null, EstadoPagamento.QUITADO, ped1, 6);
ped1.setPagamento(pagto1);

Pagamento pagto2 = new PagamentoComBoleto(null, EstadoPagamento.PENDENTE, ped2, sdf.parse("20/10/2017 00:00"), null);
ped2.setPagamento(pagto2);

pedidoRepository.save(Arrays.asList(ped1, ped2));
pagamentoRepository.save(Arrays.asList(pagto1, pagto2));

```

Commit: ItemPedido e ItemPedidoPK

- **Classe ItemPedidoPK:**

```

@Embeddable
public class ItemPedidoPK implements Serializable {
    private static final long serialVersionUID = 1L;

    @ManyToOne
    @JoinColumn(name="pedido_id")
    private Pedido pedido;

    @ManyToOne
    @JoinColumn(name="produto_id")
    private Produto produto;

```

ATENÇÃO: no hashCode e equals, incluir ambos objetos associados que identifica o item

- **Classe ItemPedido:**

```

@Entity
public class ItemPedido {

```

```

@EmbeddedId
private ItemPedidoPK id = new ItemPedidoPK();

private Double desconto;
private Integer quantidade;
private Double preco;

public ItemPedido() {
}

public ItemPedido(Pedido pedido, Produto produto, Double desconto, Integer
quantidade, Double preco) {
    super();
    id.setPedido(pedido);
    id.setProduto(produto);
    this.desconto = desconto;
    this.quantidade = quantidade;
    this.preco = preco;
}

```

Commit: Endpoint /pedidos/{id} disponibilizado

- **Checklist:**
 - Criar PedidoService
 - Criar PedidoResource
 - Proteger contra serialização Json cíclica

Commit: Atualizacao: utilizando somente JsonIgnore

Em teste realizados, o uso de @JsonManagedReference/@JsonBackRefence apresentou alguns problemas com o envio de dados Json em requisições .

Assim, ao invés de usar @JsonManagedReference/@JsonBackRefence, vamos simplesmente utilizar o @JsonIgnore no lado da associação que não deve ser serializada. Para isto faça:

- Para cada classe de domínio:
 - Apague as anotações @JsonManagedReference existentes
 - Troque as anotações @JsonBackRefence por @JsonIgnore