

Curso: Spring Boot com Ionic - Angular - Estudo de Caso Completo

Capítulo: Implementação de modelo conceitual

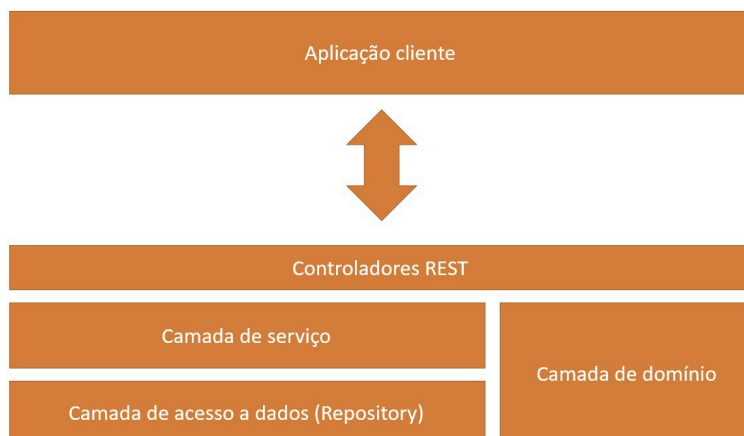
Objetivo geral:

O estudo de caso deste capítulo tem como objetivo mostrar na prática como um modelo conceitual (modelo de domínio em nível de análise) 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 seguintes tópicos:

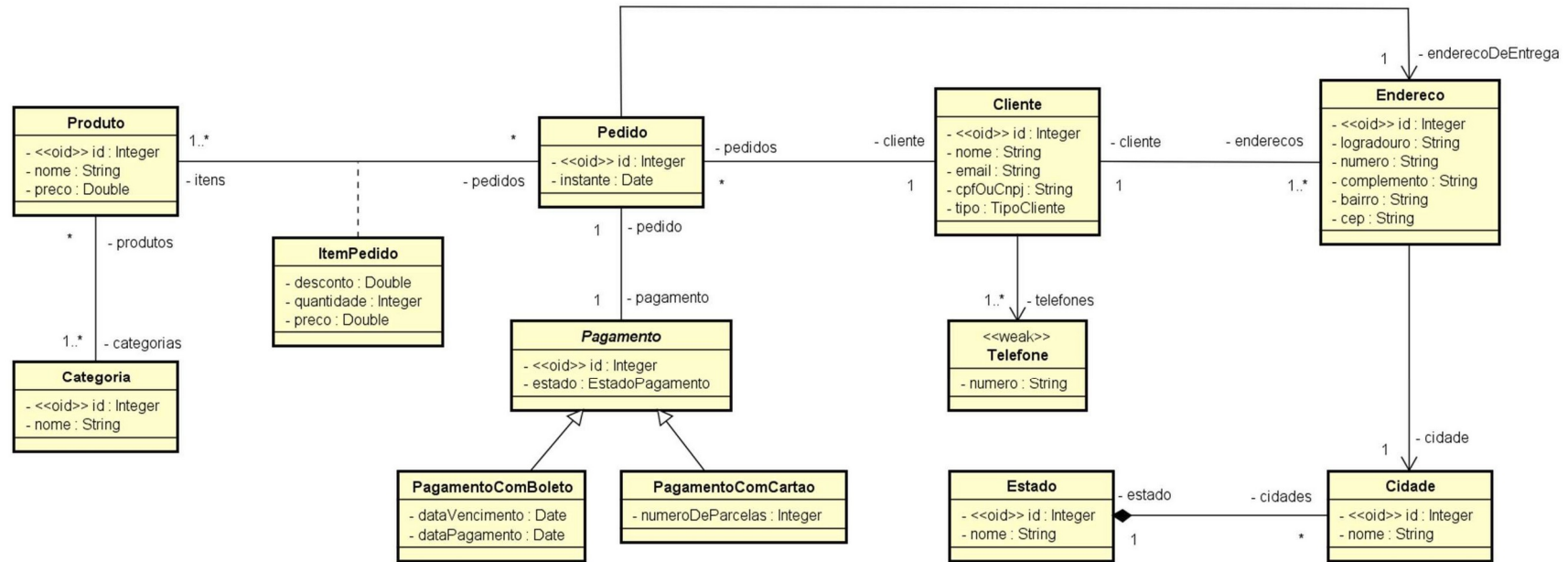
- 🕒 Leitura e entendimento do diagrama de classes
- 🕒 Leitura e entendimento do diagrama de objetos
- 🕒 Associações
 - o Um para muitos / muitos para um
 - o Um para um
 - o Muitos para muitos comum
 - o Muitos para muitos com classe de associação
 - o Bidirecionais / direcionadas
- 🕒 Conceito independente / dependente 🕒
- Classe de associação
- 🕒 Herança
- 🕒 Enumerações
- 🕒 Atributos Embedded (ItemPedidoPK)
- 🕒 Coleções ElementCollection (telefones de um cliente)

Ao longo do capítulo também vamos discutir e mostrar boas práticas de Engenharia de Software tais como desenvolvimento em camadas e tratamento de exceções. A estrutura de camadas do sistema será conforme mostrado a seguir:

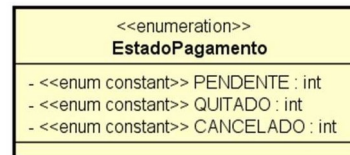
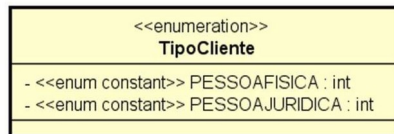


Objetivos específicos:

1) Fazer uma implementação padrão do seguinte modelo conceitual:

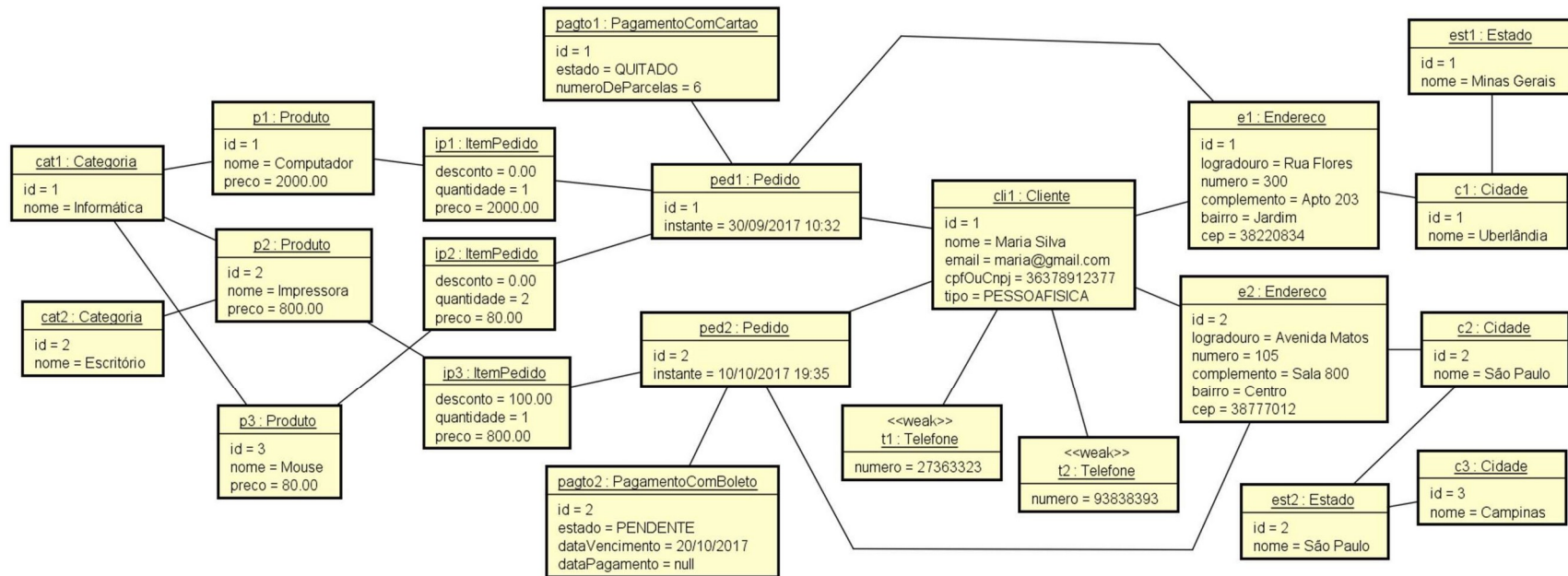


Enumerações:



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
/categorias/{id}
/clientes/{id}
/pedidos/{id}

Dados
Categoria e seus produtos
Cliente, seus telefones e seus endereços
Pedido, seu cliente, seu pagamento, seus itens de pedido, seu endereço de entrega

Criação e teste do projeto Spring Boot

Erro comum: arquivo corrompido do Maven (invalid LOC header)

Recomendação: apague os arquivos e voltar ao STS e deixar o Maven refazer o download

-

ATENÇÃO: VERSÃO DO SPRING BOOT:

Se, na criação do projeto, você escolher a versão 2.x.x, fique atento(a) às atualizações nos inícios de algumas aulas!

As atualizações serão mostradas apenas na primeira vez em que elas forem necessárias.

- ⌚ Botão direito na área da aba Package Explorer -> New -> Spring Starter Project
 - o Se não aparecer: New -> Other -> Procure

- ⌚ Opções:
 - o Name: cursomc
 - o Type: Maven
 - o Java Version: 11
 - o Group: com.nelioalves.cursomc
 - o Artifact: cursomc
 - o Version: 1.0.0-SNAPSHOT (é uma convenção do Maven)
 - o Description: Estudo de caso Java para curso de Modelagem Conceitual com UML
 - o Package: com.nelioalves.cursomc
 - o Next

- ⌚ Opções
 - o **Spring Boot Version: 2.x.x**
 - o 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"`

Testando o REST

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

```
package com.osvaldoprosper.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!";  
    }  
}
```

Testando a primeira classe de dominio - Categoria

- ⌚ Checklist para criar entidades:
 - o Atributos básicos
 - o Associações (inicie as coleções)
 - o Construtores (não inclua coleções no construtor com parâmetros)
 - o Getters e setters
 - o hashCode e equals (implementação padrão: somente id)
 - o 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;
}
```

Banco de dados H2 e criação automática da base de dados

ATUALIZAÇÃO - H2 em alguns sistemas só funcionou assim:

```
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
```

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

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

Criando repository e service para Categoria

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

Em CategoriaService, onde na aula é mostrado:

```
public Categoria find(Integer id) {
    Categoria obj = repo.findOne(id);
    return obj;
}
```

Troque pelo seguinte código (import java.util.Optional):

```
public Categoria find(Integer id) {
    Optional<Categoria> obj = repo.findById(id);
    return obj.orElse(null);
}
```

Documentação da classe Optional:

<https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>

Criando operacao de instanciacao

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

No programa principal, onde na aula é mostrado:

```
categoriaRepository.save(Arrays.asList(cat1, cat2));
```

Troque pelo seguinte código:

```
categoriaRepository.saveAll(Arrays.asList(cat1, cat2));
```

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<>();
```

Ajustes no endpoint /categorias/{id}

ATUALIZAÇÃO

Se você criou o projeto usando Spring Boot versão 2.x.x:

-

Em CategoriaService, onde na aula é mostrado:

```
public Categoria find(Integer id) {
    Categoria obj = repo.findOne(id); if
    (obj == null) {
        throw new ObjectNotFoundException("Objeto não encontrado! Id: " + id
            + ", Tipo: " + Categoria.class.getName());
    }
    return obj;
}
```

Troque pelo seguinte código:


```

public Categoria find(Integer id) {
    Optional<Categoria> obj = repo.findById(id);
    return obj.orElseThrow(() -> new ObjectNotFoundException(
        "Objeto não encontrado! Id: " + id + ", Tipo: " + Categoria.class.getName()));
}

```

- ⌚ Proteção para referência cíclica na serialização Json:

```

@JsonManagedReference
@JsonBackReference

```

- ⌚ Checklist de tratamento de exceção de id inválido:

- o Criar ObjectNotFoundException
- o Criar StandardError
- o Criar ResourceExceptionHandler

Estado e Cidade

- ⌚ Checklist para criar entidades:

- o Atributos básicos
- o Associações (inicie as coleções)
- o Construtores (não inclua coleções no construtor com parâmetros)
- o Getters e setters
- o hashCode e equals (implementação padrão: somente id)
- o 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;
}

```

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):

```

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

```

Endpoint /clientes/{id} disponivel

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

Pedido, EstadoPagamento e Pagamento

CORREÇÃO

Favor usar HH maiúsculo.

Classe principal do projeto:

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

- ⌚ 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));

```

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;
    }

```

Endpoint /pedidos/{id} disponibilizado

CORREÇÃO

Favor usar HH maiúsculo.

Classe Pedido:

`@JsonFormat(pattern="dd/MM/yyyy HH:mm")`

🕒 Checklist:

- o Criar PedidoServico
- o Criar PedidoResource
- o Proteger contra serialização Json cíclica

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`