Engenharia de Softwares Escaláveis

Design Patterns e Domain-Driven Design com Java

Agenda

Etapa 7: Projetar Softwares Usando Aggregates.

- Projeto Tático.
- Objetos de Valor.
- Entidades.

Projeto Tático

É importante identificar e modelar as **Entidades**, **Objetos de Valor** e **Agregados** que representam os conceitos fundamentais do **domínio**.

Agregados são grupos de **Entidades** e **Objetos de Valor** que são tratados como uma unidade coesa dentro do **modelo de domínio**.

Entidades são objetos identificáveis com uma **identidade única** e que mudam ao longo do tempo, enquanto **Objetos de Valor** são **objetos imutáveis** que representam valores específicos dentro do domínio.

Projetar software de forma tática no **DDD** envolve a definição e implementação de **Agregados** para garantir a consistência e a integridade dos dados dentro do sistema.

Repositórios são responsáveis por abstrair o acesso aos dados subjacentes e fornecer uma interface para recuperar e persistir objetos de domínio.

No contexto tático do DDD, é importante projetar e implementar **repositórios** que se alinhem com os conceitos do modelo de domínio e que forneçam uma interface consistente e eficaz para interagir com os dados.

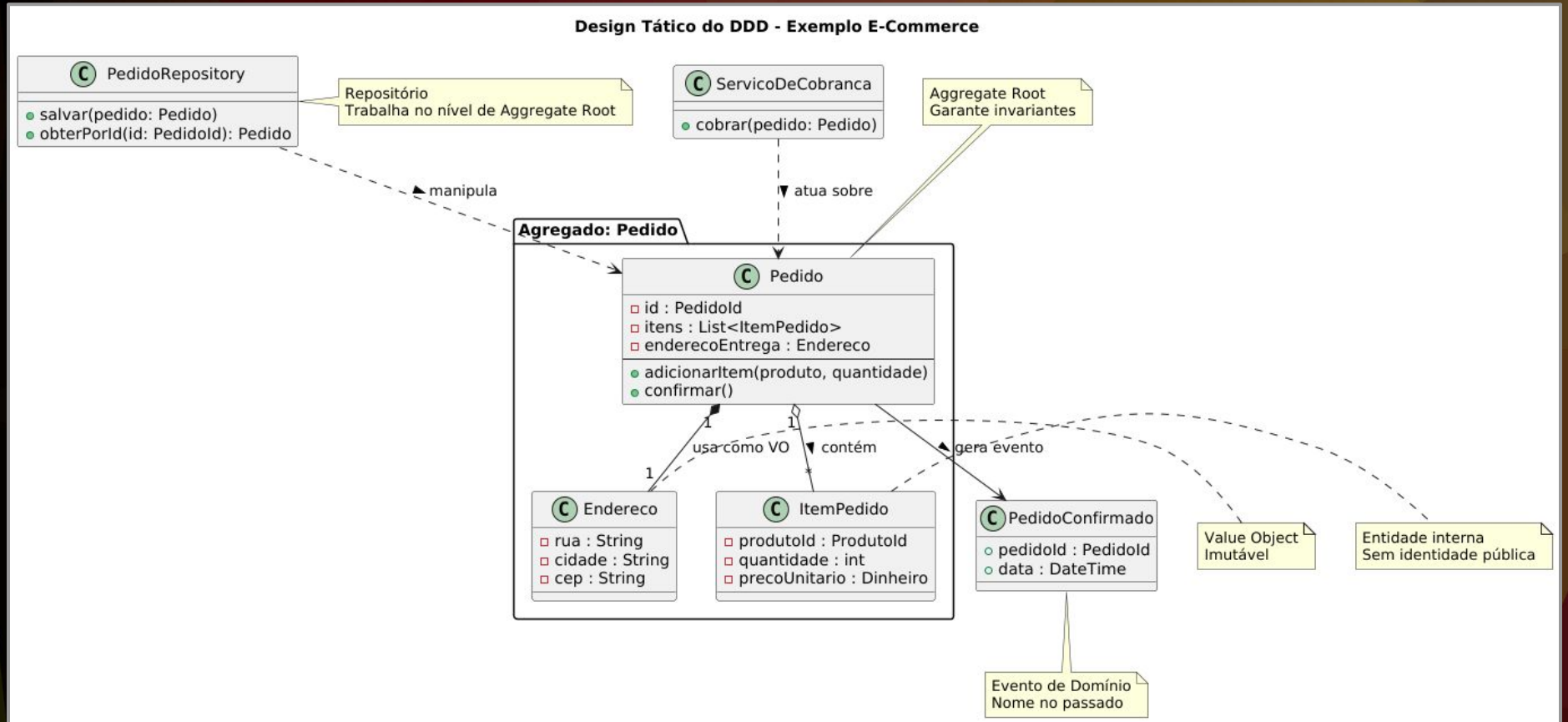
Serviços de domínio encapsulam a lógica de negócio complexa que não se encaixa naturalmente em **entidades** ou **objetos de valor** individuais.

Projetar software de forma tática no **DDD** envolve a identificação e implementação de serviços de domínio que representam as operações fundamentais do negócio e promovem a reutilização e a coesão do código.

Eventos de Domínio são **eventos** significativos que ocorrem dentro do contexto do domínio e que podem ser capturados e processados pelo sistema.

Projetar software de forma tática no **DDD** envolve a modelagem e a implementação de eventos de domínio que representam as **mudanças de estado importantes** dentro do sistema e que podem ser usados para comunicar informações entre diferentes partes do sistema.

Elemento	Descrição	Objetivo	Exemplos	Observações Importantes
Entidade	Objeto do domínio que possui **identidade única** e ciclo de vida.	Representar conceitos do domínio que mudam ao longo do tempo e precisam ser rastreados.	Cliente, Pedido, Produto, Aluno.	Identidade é mais importante que atributos. Se atributos mudam, a entidade continua a mesma.
Value Object	Objeto imutável definido **somente pelos seus valores**.	Reduzir complexidade representando conceitos simples sem identidade própria.	Endereço, CPF, Dinheiro, Email.	Deve ser imutável. Dois VOs com mesmos valores são iguais. Regras de negócio podem estar dentro de VOs.
Agregado	Conjunto de Entidades e VOs com **consistência transacional**.	Manter as **invariantes** do domínio agrupando objetos relacionados.	Pedido (Aggregate Root) + Itens do Pedido.	Define fronteiras de consistência. Cada Aggregate tem **um Aggregate Root**.
Repositório	Abstração que encapsula o acesso a coleções de Aggregates.	Permitir persistência e recuperação de Aggregates de forma transparente.	PedidoRepository, ClienteRepository.	Trabalha sempre no nível de Aggregate Root, nunca de entidades internas.
Serviço de Domínio	Operação que representa um conceito do domínio, mas que não pertence naturalmente a uma Entidade ou VO.	Capturar lógica de negócio **sem estado próprio**.	Serviço de cálculo de frete, Serviço de cobrança.	Deve ser usado apenas quando a lógica não pertence a Entidade ou VO.
Evento de Domínio	Mensagem que indica algo que **ocorreu no passado** dentro do domínio.	Propagar mudanças de estado e permitir integração entre Aggregates ou sistemas.	PedidoConfirmado, PagamentoAprovado.	Deve ter nome no passado. Pode disparar processos assíncronos ou integrações entre bounded contexts.


Design Tático do DDD - Exemplo E-Commerce
PedidoRepository
salvar(pedido: Pedido)
obterPorId(id: PedidoId): Pedido
Repositório
Trabalha no nível de Aggregate Root
ServicoDeCobranca
cobrar(pedido: Pedido)
Aggregate Root
Garante invariantes
manipula
atua sobre
Agregado: Pedido
Pedido
id : PedidoId
itens : List<ItemPedido>
enderecoEntrega : Endereco
adicionarItem(produto, quantidade)
confirmar()
1
1
usa como VO
contém
gera evento
1
*
Endereco
rua : String
cidade : String
cep : String
ItemPedido
produtoId : ProdutoId
quantidade : int
precoUnitario : Dinheiro
PedidoConfirmado
pedidoId : PedidoId
data : DateTime
Value Object
Imutável
Entidade interna
Sem identidade pública
Evento de Domínio
Nome no passado

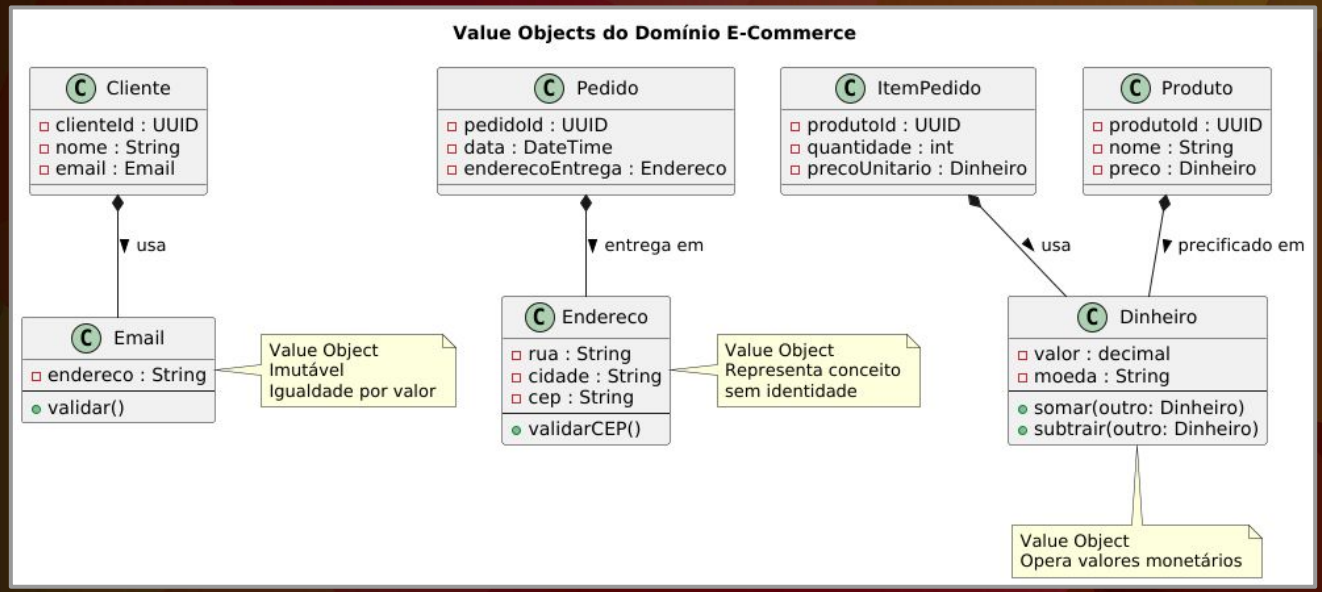
Objetos de Valor

Um **objeto de valor** modela um todo conceitual imutável.

Dentro do modelo o **objeto de valor** é apenas isso, um valor.

Ao contrário das **entidades**, ele não possui identidade única e a equivalência é determinada pela comparação dos atributos encapsulados pelos seus tipos.

Um **objeto de valor** não é uma coisa, mas é frequentemente usado para descrever, quantificar ou medir uma **entidade**.



```
class Color
{
    int _red;
    int _green;
    int _blue;
}
```

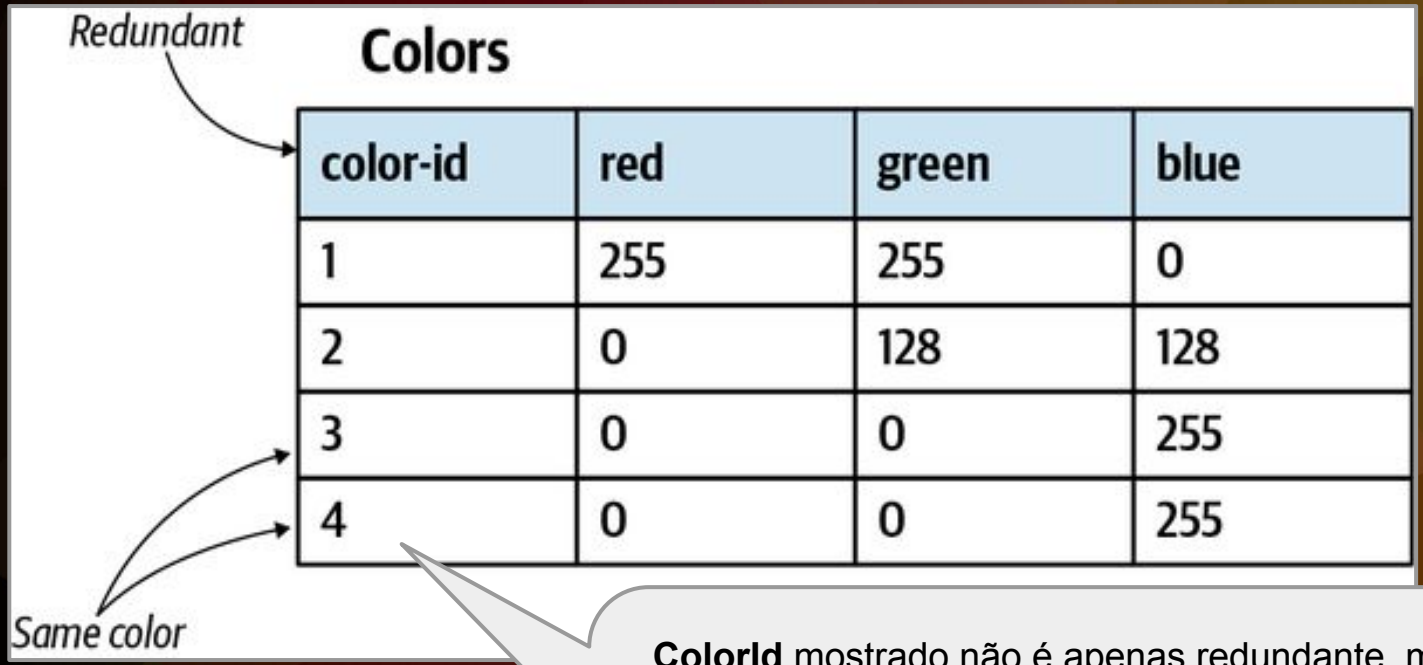
A composição dos valores dos três campos vermelho, verde e azul define uma cor.

Alterar o valor de um dos campos resultará em uma nova cor.

Não há duas cores que possam ter os mesmos valores.

Além disso, duas instâncias da mesma cor devem ter os mesmos valores.

Portanto, nenhum campo de identificação explícito é necessário para identificar cores.



ColorId mostrado não é apenas redundante, mas na verdade cria uma abertura para bugs.

Você poderia criar duas linhas com os mesmos valores de red, green, e blue, mas comparar os valores de ColorId não refletiria que esta é a mesma cor.

```
var heightMetric = Height.Metric(180);
var heightImperial = Height.Imperial(5, 3);

var string1 = heightMetric.ToString();
var string2 = heightImperial.ToString();
var string3 = heightMetric.ToImperial().ToString();

var firstIsHigher = heightMetric > heightImperial;
```

```
var phone = PhoneNumber.Parse("+359877123503");
var country = phone.Country;                          // "BG"
var phoneType = phone.PhoneType;                      // "MOBILE"
var isValid = PhoneNumber.IsValid("+972120266680"); // false
```

Objetos de Valor eliminam a necessidade de convenções — por exemplo, a necessidade de ter em mente que esta string é um e-mail e a outra string é um número de telefone — e, em vez disso, faz com que o uso do modelo de objeto seja menos propenso a erros e mais intuitivo.

```
var red = Color.FromRGB(255, 0, 0);
var green = Color.Green;
var yellow = red.MixWith(green);
var yellowString = yellow.ToString();
```

Como uma alteração em qualquer um dos campos de um objeto de valor resulta em um valor diferente, os objetos de valor são implementados como objetos imutáveis.

Uma alteração em um dos campos do objeto de valor cria conceitualmente um valor diferente – uma instância diferente de um objeto de valor.

```
public class Color
{
    public readonly byte Red;
    public readonly byte Green;
    public readonly byte Blue;

    public Color(byte r, byte g, byte b)
    {
        this.Red = r;
        this.Green = g;
        this.Blue = b;
    }

    public Color MixWith(Color other)
    {
        return new Color(
            r: (byte) Math.Min(this.Red + other.Red, 255),
            g: (byte) Math.Min(this.Green + other.Green, 255),
            b: (byte) Math.Min(this.Blue + other.Blue, 255)
        );
    }

    ...
}
```

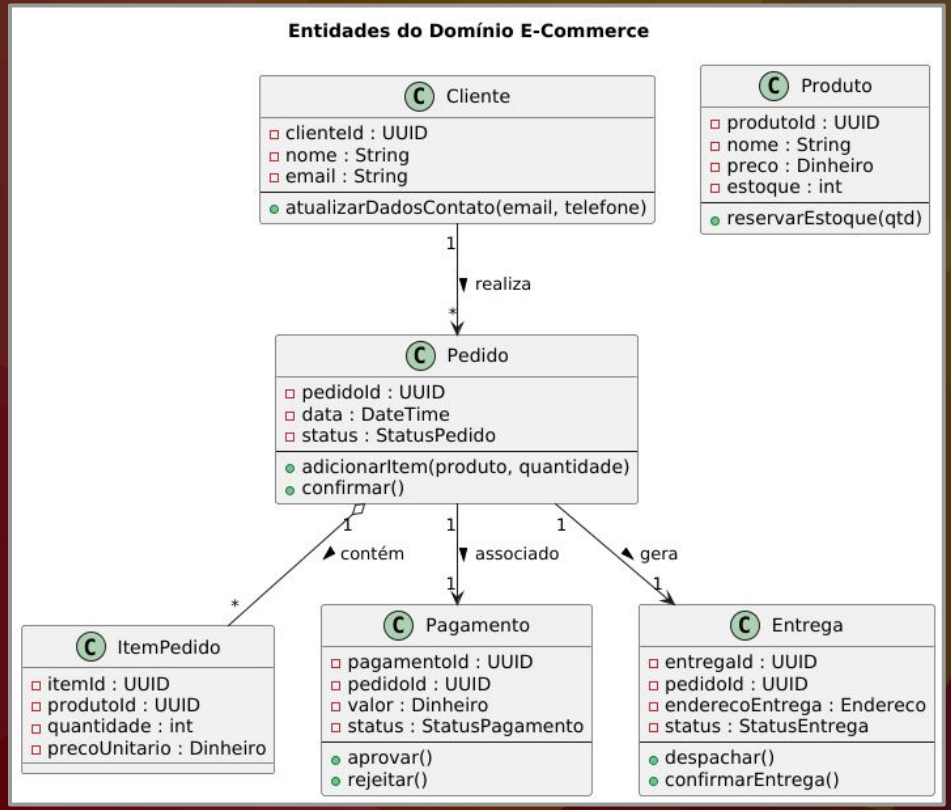
Entidades

Uma **entidade** modela uma coisa individual.

Cada **entidade** tem uma identidade única, na medida em que é possível distinguir a sua individualidade entre todas as outras **entidades** do mesmo tipo ou de tipos diferentes.

Muitas vezes, uma **entidade** poderá ter seu estado alterado com o tempo, mas não sua identidade.

A principal coisa que separa uma **entidade** de outras ferramentas de modelagem é sua singularidade – sua individualidade.



Pessoa.java

```java
package br.edu.infnet.domain;

import ...

public class Pessoa {

    private final String id;       // Identidade única da Entidade
    private String nome;
    private String email;
    private LocalDate dataNascimento;
    private int idade;

    public Pessoa(String nome, String email, LocalDate dataNascimento) {
        this.id = UUID.randomUUID().toString(); // gera identidade única
        this.nome = nome;
        this.email = email;
        this.dataNascimento = dataNascimento;
        this.idade = calcularIdade(dataNascimento);
    }
```

O objeto de valor deixa a intenção clara, mesmo com nomes de variáveis mais curtos.

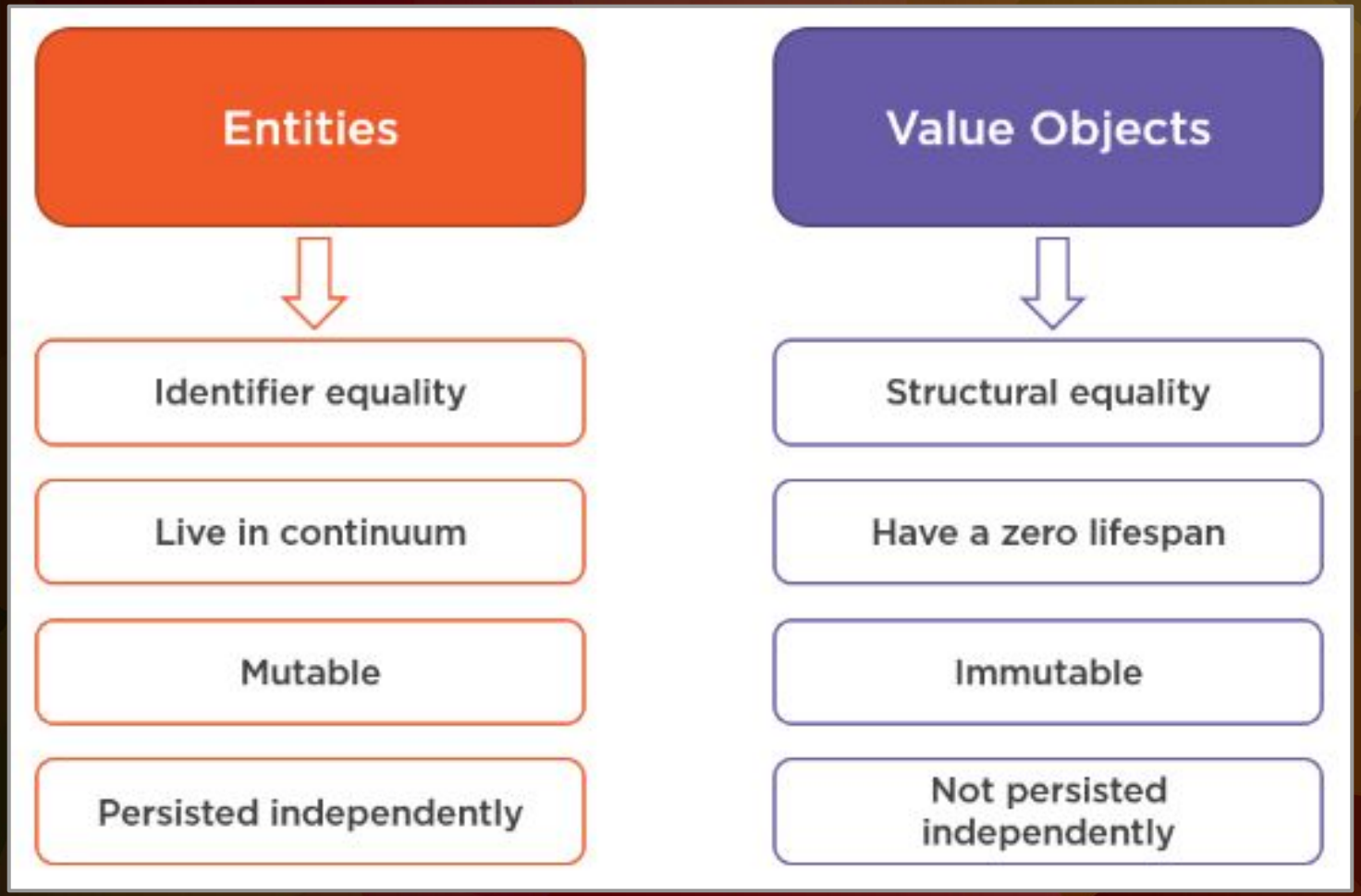
Não há necessidade de validar os valores antes da atribuição, pois a lógica de validação reside nos próprios objetos de valor.

Os objetos de valor brilham mais quando centralizam a lógica de negócios que manipula os valores.

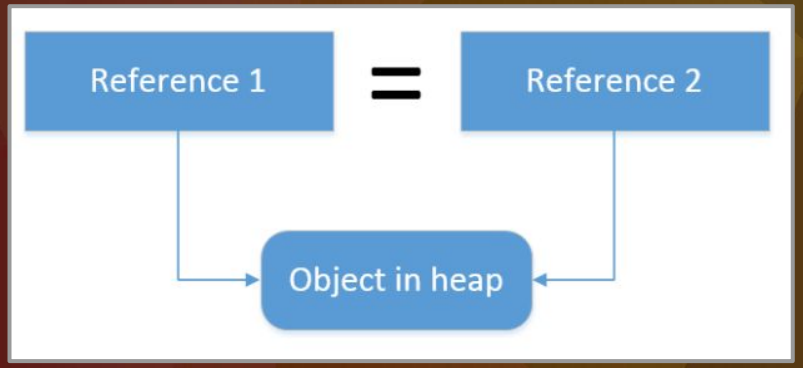
Pessoa.java

```java
package br.edu.infnet.domain;

import ...

public class Pessoa {

    private final String id; // Identidade única da Entidade
    private NomeCompleto nome;
    private Email email;
    private DataDeNascimento dataDeNascimento;

    public Pessoa(NomeCompleto nome, Email email, DataDeNascimento dataDeNascimento) {
        this.id = UUID.randomUUID().toString();
        this.nome = Objects.requireNonNull(nome);
        this.email = Objects.requireNonNull(email);
        this.dataDeNascimento = Objects.requireNonNull(dataDeNascimento);
    }
```

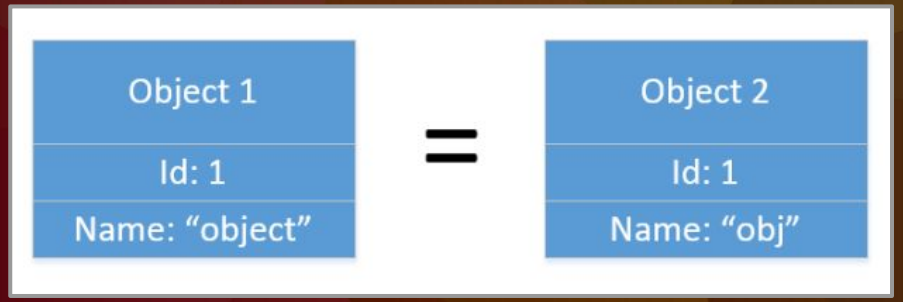
Aspecto	Pessoa com Primitivos	Pessoa com Value Objects
Validação	Dispersa, feita dentro da Entidade.	Encapsulada dentro dos VOs (`new Email("x@x.com")` já garante validade).
Expressividade	Código menos expressivo, apenas tipos genéricos.	Código mais rico e próximo da linguagem ubíqua do domínio.
Reuso de regras	Difícil, validações precisam ser copiadas em cada uso.	Regras centralizadas dentro dos VOs, podem ser reutilizadas em qualquer Entidade.
Mutabilidade	Fácil de alterar valores sem garantir consistência.	VOs são imutáveis → consistência natural.
Igualdade	Igualdade da Entidade depende apenas do `id`.	Igualdade dos VOs é por valor (dois `Email` iguais são o mesmo objeto conceitualmente).
Proteção contra erros	Baixa → qualquer String pode ser usada como email, mesmo inválida.	Alta → só um `Email` válido pode existir, evitando inconsistência no modelo.


Entities
Identifier equality
Live in continuum
Mutable
Persisted independently
Value Objects
Structural equality
Have a zero lifespan
Immutable
Not persisted independently

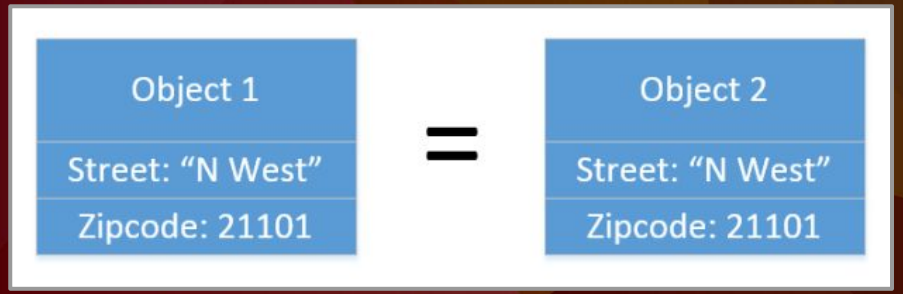
Igualdade de Referência



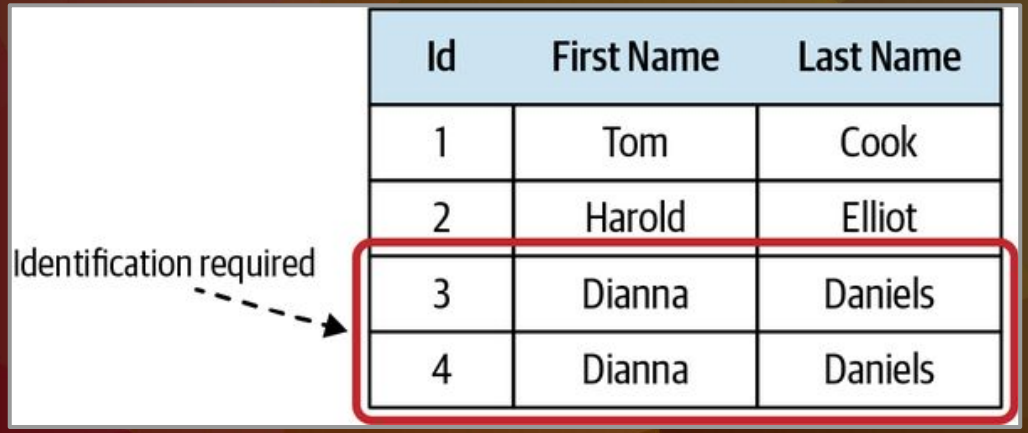
Igualdade de Identificadores
Entidades



Igualdade Estrutural
Objetos de Valor



```
class Person
{
    public readonly PersonId Id;
    public Name Name { get; set; }

    public Person(PersonId id, Name name)
    {
        this.Id = id;
        this.Name = name;
    }
}
```



Uma **entidade** é o oposto de um **objeto de valor**.

Isso requer um campo de identificação explícito para distinguir entre as diferentes instâncias da entidade.

O campo de identificação Id do tipo PersonId. PersonId é um objeto de valor e pode usar qualquer tipo de dados subjacente que atenda às necessidades do domínio de negócios.