

Engenharia de Softwares Escaláveis

Domain-Driven Design (DDD) e Arquitetura de
Softwares Escaláveis com Java

Agenda

Etapas 1: Aplicando Aggregates e Bounded Contexts.

- Design Tático Orientado a Domínio.
- Projeto de Microsserviços.
- Value Object.



Design Tático Orientado a Domínio

Design Tático é um conjunto de padrões de design e blocos de construção que você pode usar para projetar sistemas orientados a domínio.

Mesmo para projetos que não são orientados a domínio, você pode se beneficiar do uso de alguns dos padrões táticos de DDD.

Design Tático visa refinar o modelo de domínio a um estágio em que ele possa ser convertido em código funcional.

Design Tático é muito mais prático e mais próximo do código real do que o **Design Estratégico** orientado a domínio.

Design Estratégico lida com elementos abstratos, enquanto o **Design Tático** lida com classes e módulos.

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.

Projeto de Microserviços



**Pet
Friends**

Atendimento
ao Cliente

Atendimento
ao
Franqueado

Produtos

Serviços
Agendados

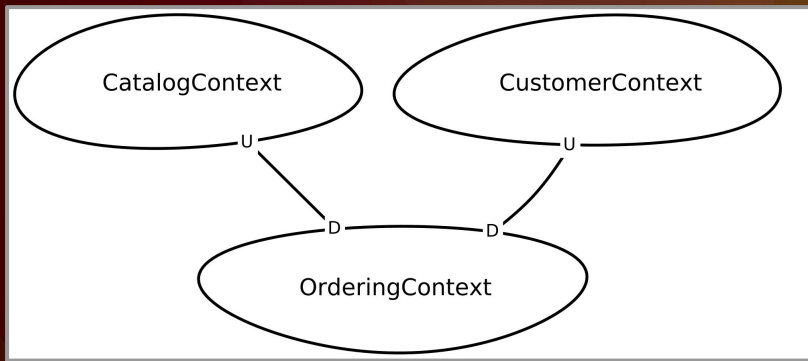
Venda

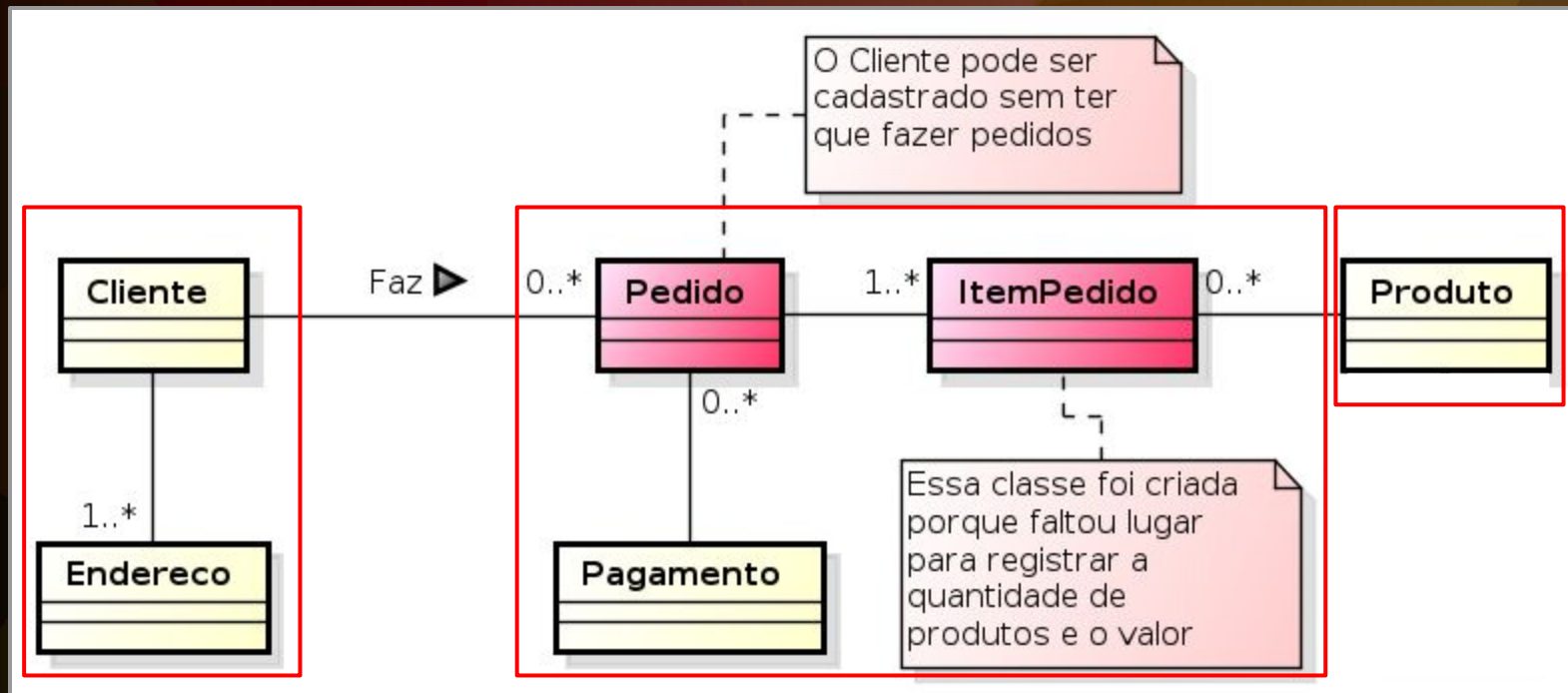
Assinatura
de Ração

Veterinário

Tosa e
Banho

Passeio

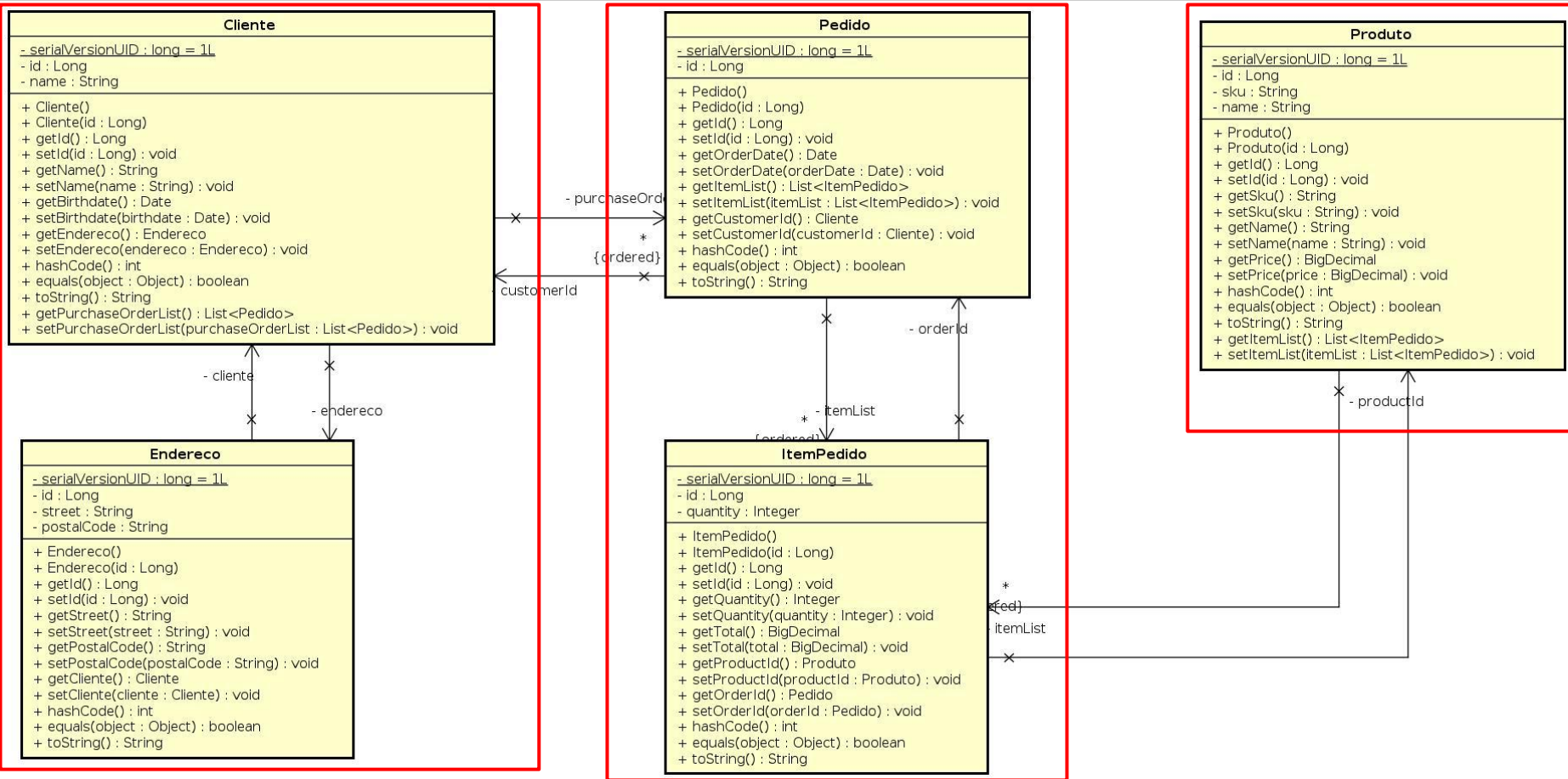




1

2

3





Value Object

Value Object modela um todo conceitual imutável.

Dentro do modelo, o **Value Object** é apenas isso, um valor.

Ao contrário de uma **Entity**, ele não tem uma identidade única, e a equivalência é determinada pela comparação dos atributos encapsulados pelo seu tipo.

Além disso, um **Value Object** não é uma coisa, mas é frequentemente usado para descrever, quantificar ou medir uma **Entity**.

1

ValorMonetario.java x

Source

History

```
1 package br.edu.infnet.valueObject;
```

```
2
3 import java.io.Serializable;
4 import java.math.BigDecimal;
5 import java.math.RoundingMode;
6 import java.util.Objects;
```

```
7
8 public class ValorMonetario implements Serializable {
```

```
9
10     private final BigDecimal quantia;
```

```
11
12     public ValorMonetario(BigDecimal quantia) {
```

```
13         if (quantia == null || quantia.signum() < 0) {
14             throw new IllegalArgumentException("Valor Monetário não pode ser negativo");
15         }
```

```
16         this.quantia = quantia.setScale(2, RoundingMode.HALF_UP);
17     }
```

```
18
19     public BigDecimal getQuantia() {
20         return this.quantia;
21     }
```

Aqui podemos definir a "natureza" do value object

2

```
ValorMonetario.java x
Source History
23 public ValorMonetario somar(ValorMonetario outro) {
24     if (outro == null) {
25         throw new IllegalArgumentException("Outro valor não pode ser nulo");
26     }
27     return new ValorMonetario(this.quantia.add(outro.getQuantia()));
28 }
29
30 public ValorMonetario subtrair(ValorMonetario outro) {
31     if (outro == null) {
32         throw new IllegalArgumentException("Outro valor não pode ser nulo");
33     }
34     return new ValorMonetario(this.quantia.subtract(outro.getQuantia()));
35 }
36
37 @Override
38 public boolean equals(Object objeto) {
39     final ValorMonetario outro = (ValorMonetario) objeto;
40     return Objects.equals(this.quantia, outro.getQuantia());
41 }
42 }
```



```
ValorMonetarioTest.java x
Source History
1 package br.edu.infnet.valueObject;
2
3 import java.math.BigDecimal;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 public class ValorMonetarioTest {
7
8     @org.junit.jupiter.api.Test
9     public void testSomar() {
10         System.out.println("Soma de valores monetários");
11         ValorMonetario valor1 = new ValorMonetario(new BigDecimal(200.34));
12         ValorMonetario valor2 = new ValorMonetario(new BigDecimal(100));
13         ValorMonetario esperado = new ValorMonetario(new BigDecimal(300.34));
14         ValorMonetario resultado = valor1.somar(valor2);
15         assertEquals(esperado.getQuantia(), resultado.getQuantia());
16     }
17
18     @org.junit.jupiter.api.Test
19     public void testSubtrair() {
20         System.out.println("Subtração de valores monetários");
21         ValorMonetario valor1 = new ValorMonetario(new BigDecimal(200.34));
22         ValorMonetario valor2 = new ValorMonetario(new BigDecimal(100));
23         ValorMonetario esperado = new ValorMonetario(new BigDecimal(100.34));
24         ValorMonetario resultado = valor1.subtrair(valor2);
25         assertEquals(esperado.getQuantia(), resultado.getQuantia());
26     }
27 }
```

3



4

Email.java x

Source History

```
1 package br.edu.infnet.valueObject;
2
3 import java.util.Objects;
4 import org.apache.commons.validator.routines.EmailValidator;
5
6 public class Email {
7
8     private static final EmailValidator validator = EmailValidator.getInstance();
9     private final String email;
10
11     public Email(String email) {
12         if (!validator.isValid(email)) {
13             throw new IllegalArgumentException("Endereço de email inválido");
14         }
15         this.email = email;
16     }
17
18     public Email alterar(String value) {
19         return new Email(value);
20     }
21
22     @Override
23     public boolean equals(Object o) {
24         Email that = (Email) o;
25         return Objects.equals(email, that.email);
26     }
27 }
```

4

Email.java x pom.xml [DesignTatico] x

Source Graph Effective History

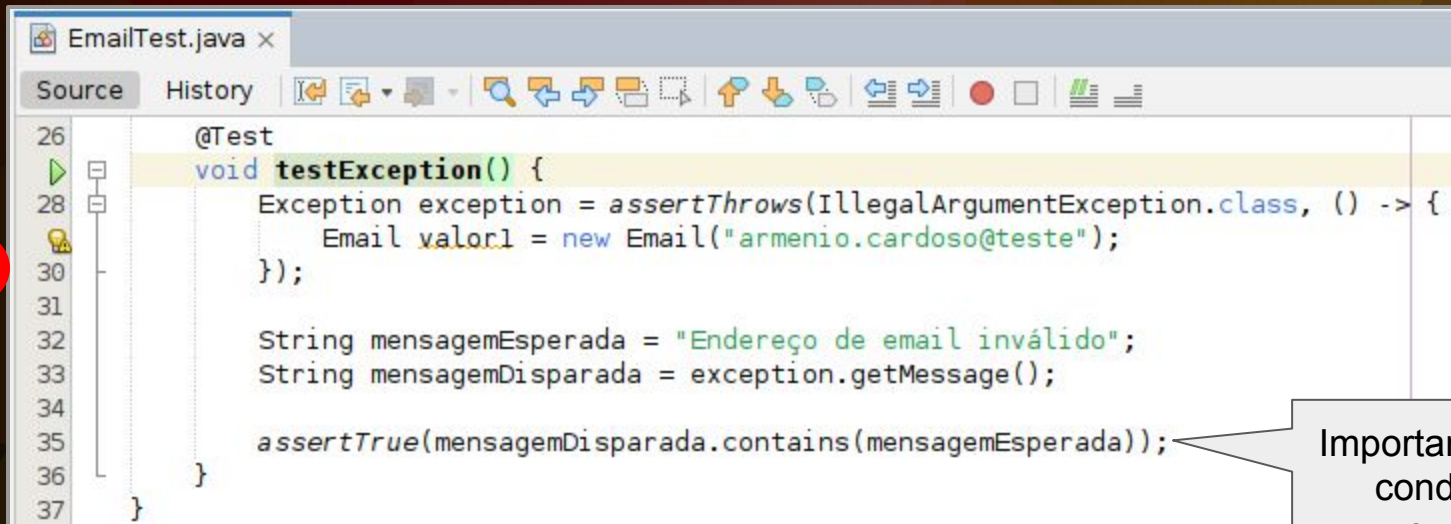
```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <groupId>br.edu.infnet</groupId>
6     <artifactId>DesignTatico</artifactId>
7     <version>1.0-SNAPSHOT</version>
8     <packaging>jar</packaging>
9     <dependencies>
10         <dependency>
11             <groupId>commons-validator</groupId>
12             <artifactId>commons-validator</artifactId>
13             <version>1.9.0</version>
14         </dependency>
15     </dependencies>
16 </project>
```

5

```
EmailTest.java x
Source History
1 package br.edu.infnet.valueObject;
2
3 import org.junit.jupiter.api.Test;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 public class EmailTest {
7
8     @Test
9     public void testAlterar() {
10         Email valor1 = new Email("armenio.cardoso@prof.infnet.edu.br");
11         Email esperado = new Email("armeniotorres@gmail.com");
12         Email resultado = valor1.alterar("armeniotorres@gmail.com");
13         System.out.println("Teste de troca de emails");
14         assertEquals(esperado, resultado);
15     }
16
17     @Test
18     public void testEquals() {
19         Email valor1 = new Email("armenio.cardoso@prof.infnet.edu.br");
20         Email valor2 = new Email("armenio.cardoso@prof.infnet.edu.br");
21         System.out.println("Teste de igualdade de emails");
22         boolean resultado = valor1.equals(valor2);
23         assertEquals(true, resultado);
24     }
25 }
```



6



The screenshot shows an IDE window titled "EmailTest.java". The editor displays a Java test method named `testException()` starting at line 26. The code uses `assertThrows` to expect an `IllegalArgumentException` from a lambda expression that creates an `Email` object with the address "armenio.cardoso@teste". The expected error message is "Endereço de email inválido". A lightbulb icon is present next to line 29. The code ends at line 37.

```
26     @Test
27     void testException() {
28         Exception exception = assertThrows(IllegalArgumentException.class, () -> {
29             Email valor1 = new Email("armenio.cardoso@teste");
30         });
31
32         String mensagemEsperada = "Endereço de email inválido";
33         String mensagemDisparada = exception.getMessage();
34
35         assertTrue(mensagemDisparada.contains(mensagemEsperada));
36     }
37 }
```

Importante testar as condições de exceções

Builder

O padrão do construtor ajuda a separar a construção de um objeto complexo a partir de sua representação de código para que o mesmo processo de composição possa ser reutilizado para criar diferentes configurações de um tipo de objeto.

A principal motivação por trás do padrão construtor é construir instâncias complexas sem poluir o construtor.

Ajuda a separar ou mesmo dividir o processo de criação em etapas específicas.

A composição dos objetos é transparente para o cliente e permite a criação de diferentes configurações do mesmo tipo.

```
Telefone.java x
Source History
1 package br.edu.infnet.valueObject;
2
3 public class Telefone {
4
5     private int ddd;
6     private int numero;
7     private TipoFone tipo;
8
9     private Telefone() {
10    }
11
12    public int getDdd() {
13        return ddd;
14    }
15
16    public int getNumero() {
17        return numero;
18    }
19
20    public TipoFone getTipo() {
21        return tipo;
22    }
23
24    public enum TipoFone {
25        RESIDENCIAL, CELULAR, COMERCIAL
26    }
```

7

Source History

History

```
public boolean equals(Object objeto) {
```

```
return false;
```

i

```

return false:

```

f

```
if (this ddd != outro ddd) {
```

```

if (this.add != 0)
    return false;

```

5

```
if (this.numero != 0)
    return false;
```

2

}

```
if (this.tipo != outro.tipo) {
```

10

}

```
return true;
```

1

Telephone.java x

Source History

```
49 public static class Builder {
50
51     private final Telefono telefono = new Telefono();
52
53     public Builder addDdd(int ddd) {
54         if (ddd <= 0) {
55             throw new IllegalArgumentException("DDD inválido");
56         }
57         telefono.ddd = ddd;
58         return this;
59     }
60
61     public Builder addNumero(int numero) {
62         if (numero <= 0) {
63             throw new IllegalArgumentException("DDD inválido");
64         }
65         telefono.numero = numero;
66         return this;
67     }
68
69     public Builder addTipoFone(TipoFone tipoFone) {
70         telefono.tipo = tipoFone;
71         return this;
72     }
73
74     public Telefono build() {
75         return this.telefono;
76     }
77 }
78 }
```


10

```
TelephoneTest.java x
Source History
1 package br.edu.infnet.valueObject;
2
3 import org.junit.jupiter.api.Test;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 public class TelefoneTest {
7
8     @Test
9     public void testTelefone() {
10         System.out.println("Teste Telefone");
11         Telefone tel1 = new Telefone.Builder()
12             .addDdd(21)
13             .addNumero(21228800)
14             .addTipoFone(Telefone.TipoFone.COMERCIAL).build();
15         Telefone tel2 = new Telefone.Builder()
16             .addDdd(21)
17             .addNumero(21228800)
18             .addTipoFone(Telefone.TipoFone.COMERCIAL).build();
19         assertEquals(tel1, tel2);
20     }
21 }
```



Modelo de Domínio Anêmico

Existem objetos, muitos nomeados após os substantivos no espaço de domínio, e esses objetos estão conectados com os relacionamentos e estruturas ricos que os modelos de domínio verdadeiros têm.

O problema vem quando você olha para o comportamento e percebe que quase não há comportamento nesses objetos, **tornando-os pouco mais do que sacos de getters e setters.**

```
Endereco.java x
Source History
1 package br.edu.infnet.valueObject;
2
3 public class Endereco {
4
5     private String logradouro;
6     private String numero;
7     private String complemento;
8     private String bairro;
9     private String cidade;
10    private String cep;
11
12    public String getLogradouro() {
13        return logradouro;
14    }
15
16    public void setLogradouro(String logradouro) {
17        this.logradouro = logradouro;
18    }
19
20    public String getNumero() {
21        return numero;
22    }
23
24    public void setNumero(String numero) {
25        this.numero = numero;
26    }
27
28    public String getComplemento() {
29        return complemento;
30    }
}
```

```
viacep.com.br/ws/01001000 x +
viacep.com.br/ws/01001000/json/
Estilos de formatação ✓
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "unidade": "",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

Vamos Exercitar?

Como ficaria essa classe
“anêmica” implementada como
value object com builder?

