



Curso de Java

aula 04

Prof. Anderson Henrique



Programação Orientada a Objetos?



DIFICULDADES

- Complexidade no aprendizado em comparação com a programação estruturada
- Seus conceitos são de difícil compreensão



BENEFÍCIOS

- Mais fácil descrever o mundo real através dos objetos
- O encapsulamento facilita a manutenção do código
- Maior facilidade para reutilização de código







Conceitos de Orientação a Objetos

- POO – Programação Orientada a Objetos;
- Objetivo – Aproximar o mundo digital do mundo real;
- Como era? Programação Baixo Nível-Programação Linear-Programação Estruturada - Programação Modular - POO
- Quem criou? Alan Kay 1970 (Era formado em Matemática e Biologia);
- Programação Estruturada – Programação Modular (Dados Globais) -> Procedimentos | Dados de Objetos -> Métodos



Vantagens

- Confiável -> O isolamento entre as partes gera software seguro. Ao alterar uma parte, nenhuma outra é afetada;
- Oportuno -> Ao dividir tudo em partes, várias delas podem ser desenvolvidas em paralelo;
- Manutenível -> Atualizar um software é fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto;
- Extensível -> O software não é estático. Ele deve crescer para permanecer útil;
- Reutilizável -> Podemos usar objetos de um sistema que criamos em outro sistema futuro;

- Natural -> Mais fácil de entender. Você se preocupa mais na funcionalidade do que nos detalhes de implementação;

Classe



É uma estrutura estática utilizada para descrever objetos mediante atributos e métodos. A classe é um Modelo (Model) ou Template para criação desses objetos. Ela é representada pelo operador (class).



Atributos

São as características que definem o que o objeto é, os atributos são as variáveis da classe, poderão armazenar valores que serão manipulados através dos métodos da classe

Ex.: **atributo**: String nome

Pode ser manipulado através do **método**:
alterarNome(String nome)



Métodos

São as funcionalidades da classe, servem para manipular os dados de um ou mais atributos.

Os métodos podem ser declarados na classe ou podem ser declarados e implementados

Ex.: **void** Crescer(float centimetros);

void Crescer(float centimetros){

aqui vem a implementação...

}



O que é um objeto?

- Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas **características, comportamentos e estado atual**.
- Objeto caneta (**Coisas que eu tenho** [Modelo, Cor, Ponta, Carga, Tampada], **Coisas que eu faço** [Escrever, Rabiscar, Pintar, Tampar, Destampar], **Como eu estou agora** [50% de Carga, Destampada, Escrevendo]).

Classe Caneta

Atributos

modelo: Caractere

cor: Caractere

ponta: Real

carga: inteiro

tampada: lógico

Métodos

Método rabiscar()

FimMétodo

Método tampar()

FimMétodo

FimClasse





Como podemos criar objeto?



Classe

```
c1 = new Caneta  
c1.cor = "Azul"  
c1.ponta = 0.5  
c1.tampada = false
```

instanciar

```
c2 = new Caneta  
c2.cor = "Vermelho"  
c2.ponta = 1.0  
c2.tampada = true
```



Objeto

- **Classe** -> Define os atributos e métodos comuns que serão compartilhados por um objeto.
- **Objeto** -> É a instância de uma classe.

Linguagem de Modelagem Unificada (UML)

Não veremos de forma aprofundada sobre essa temática, no entanto, precisamos utilizar o Diagrama de Classes.

Caneta

- + modelo: String
- + cor: String
- + ponta: float
- + carga: int
- + tampada: boolean

- + escrever()
- + rabiscar()
- + pintar()
- + tampar()
- + destampar()

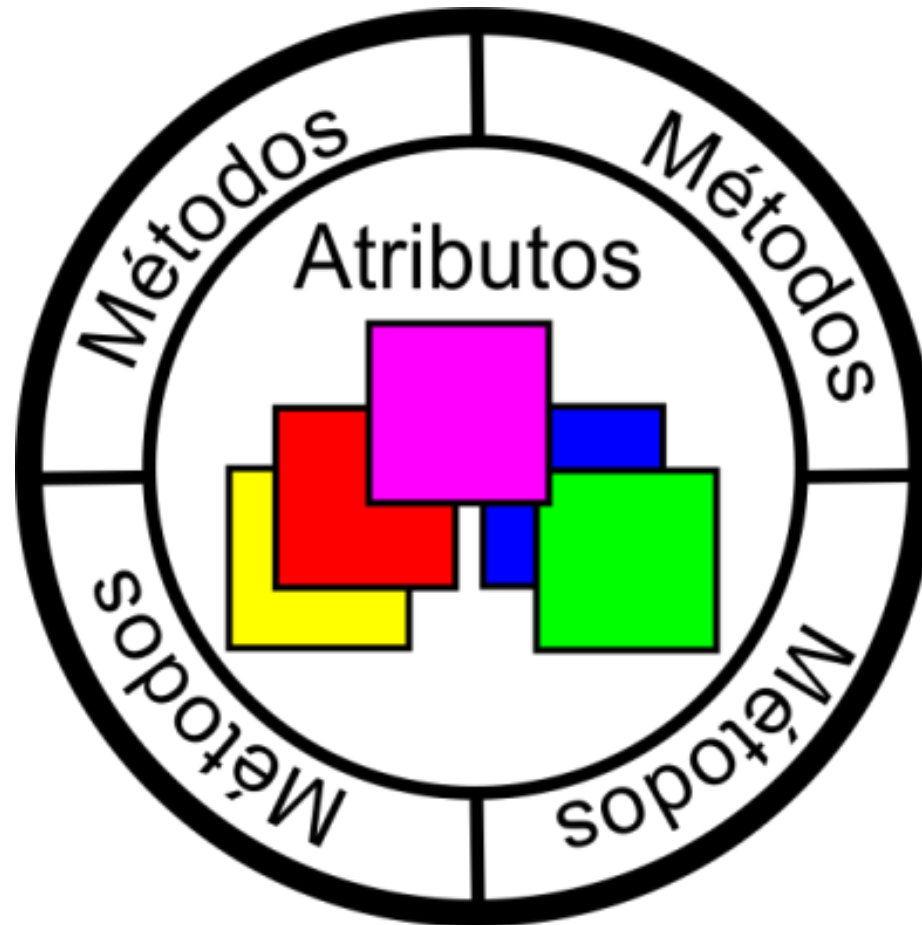
Modificadores de Visibilidade

Indicam o nível de acesso aos componentes internos de uma classe (atributos e métodos).

Na UML, estes modos de visibilidade são representados da seguinte forma:

- + public (público)
- private (privado)
- # protected (protegido)

Podemos definir a visibilidade dos atributos e dos métodos de um objeto. A visibilidade define a forma como esses atributos devem ser acessados



Visibilidade	Descrição
Private (-)	Membros declarados como private somente podem ser acessados dentro da própria classe em que foram declarados.
Protected (#)	Membros declarados como protected somente podem ser acessados dentro da própria classe e a partir de classes descendentes (classes-filha), mas não poderão ser acessados a partir do programa que faz uso dessa classe.
Public (+)	Membros declarados como public poderão ser acessados livremente a partir da própria classe, a partir de classes descendentes e a partir do programa que faz uso dessa classe.

Tabela dos modificadores de acesso

	private	default	protected	public
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasses)	não	não	sim	sim
pacotes diferentes (sem subclasses)	não	não	não	sim

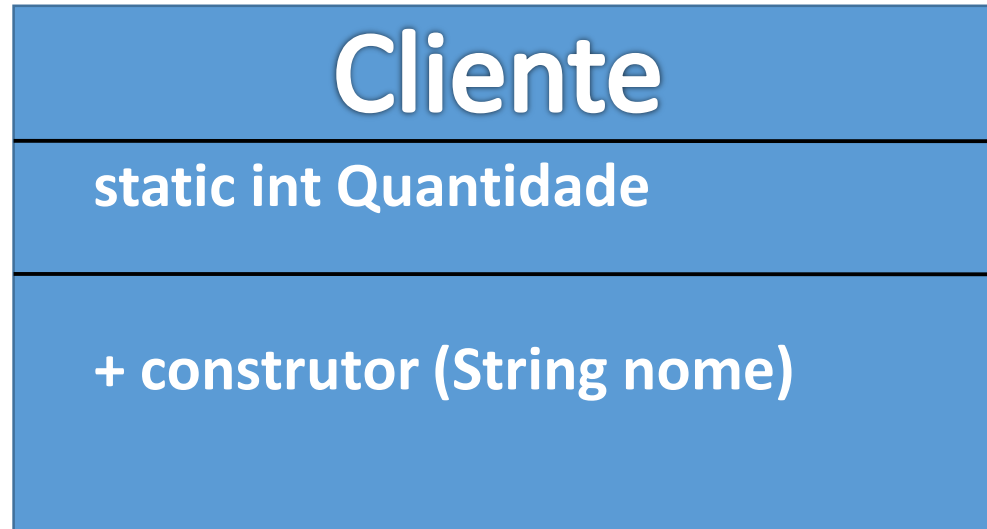
Constantes

Declaramos uma constante de classe utilizando o modificador (**final**).

```
private final long NUMERO_ATRIBUTO =  
    547L;  
private final double NUMERO_ATRIBUTO =  
    522.80;
```

Atributos estáticos

São atributos de um objeto. São dinâmicos como os atributos de um objeto, mas estão relacionados à classe. Como a classe é a estrutura comum a todos os objetos dela derivados, são compartilhados entre todos os objetos de uma mesma classe



→ **Nome da Classe**

→ **Propriedade**

→ **Construtor**

Métodos estáticos

Podem ser invocados diretamente da classe, sem a necessidade de instanciar um objeto para isso. São limitados a chamarem outros métodos estáticos da classe ou utilizar apenas atributos estáticos

Vendedor

static String Nome
static double Salario

+ static void Exibir ()

→ Nome da Classe

→ Propriedades

→ Método

Método set

Intercepta a atribuição de valores a propriedades de um objeto. Sempre que for atribuído um valor a uma propriedade do objeto, automaticamente esta atribuição passa pelo método `set()`, o qual recebe o nome da propriedade e o valor a ser atribuído.

Ex.: **`public void setNome(String nome)`**

Método get

Intercepta as requisições de propriedades do objeto. Sempre que for requisitada uma propriedade, automaticamente passará pelo método `get()`, o qual recebe o nome da propriedade.

Ex.: **`public String getNome(){return;}`**

Pessoa

+ Nome: (string)
+ Idade: (int)
+ Peso: (float)
+ Altura: (float)
+ Escolaridade: (string)
+ Salario: (double)

+ alteraNome (String nome)
+ Envelhecer (int anos)
+ alteraPeso (float quilogramas)
+ Crescer (float centimetros)
+ Formar (String titulacao)
+ alteraSalario (double quantia)
+ Exibir ()

→ **Nome da Classe**

→ **Atributos**

→ **Métodos**



Produto

+ Cod: (int)
+ Descricao: (string)
+ Fabricante: (string)
+ Quantidade: (int)
+ Preco: (float)

+ venderProduto (int quantidade)
+ reporEstoque (int quantidade)
+ alteraPreco (float valor)
+ Exibe ()

→ **Nome da Classe**

→ **Propriedades**

→ **Métodos**



Construtor

É um método especial utilizado para definir o comportamento inicial de um objeto, ou seja, o comportamento no momento da criação. É um método executado automaticamente.

Pode ser gerado no netbeans com o atalho
(ALT + Insert)



Cachorro

+ Coleira (int)
+ Nome (string)
+ Idade (int)
+ Raca (string)

+ construtor () { ... }

+ Latir ()
+ Exibe ()

→ **Nome da Classe**

→ **Propriedades**

→ **Construtor**

→ **Métodos**



Herança

A possibilidade de reutilizar partes de código já definidas é o que nos dá maior agilidade no dia-a-dia, além de eliminar a necessidade de eventuais duplicações ou reescritas de código.

Os atributos e os métodos declarados na super-classe são herdadas em todas as subclasses que dela se estendem



Conta

- + Agencia: (string)
- + CC: (string)
- + Titular: (string)
- + DataCriacao: (date)
- + Senha: (string)
- + Saldo: (float)
- + Cancelada: (boolean)

- + Retirar (double quantia)
- + Depositar (double quantia)
- + obterSaldo ()

ContaCorrente

- + Limite: (float)

- + Retirar (double quantia)



Polimorfismo

Classes derivadas de uma mesma super-classe que tenham métodos sobrescritos (`@Override`), mas comportamentos diferentes, redefinidos em cada uma das subclasses.

O método `Retirar()` na classe `Conta` somente efetua o saque, na `ContaCorrente` verifica se está dentro do limite



Abstração

Classes estruturais, que na hierarquia de classes servem de base para outras. São classes que nunca serão instanciadas na forma de objetos, somente as suas subclasses serão. Marcamos essas classes como abstratas (`abstract`). No exemplo da classe `Conta`, esta jamais poderá ser instanciada, então a marcamos como class **`abstract{ }`**



Método abstrato

São métodos declarados na classe abstrata sendo somente implementados nas subclasses. Métodos abstratos somente podem ser declarados em classe abstrata, sua implementação nas subclasses se torna obrigatória.



Classe final

A classe final não pode ser uma superclasse, ou seja, não pode ser base em uma estrutura de herança. Se definirmos uma classe como final pelo modificador **final**, ela não poderá mais ser especializada (herdada) em subclasses.



Método final

Um método final não pode ser sobrescrito, ou seja, não pode ser redefinido nas subclasses. Trabalhando com herança, os métodos definidos como finais não poderão sofrer o (`@Override`).



Interfaces

A POO baseia-se fortemente na interação de classes e objetos. Um objeto deve conhecer quais são as funcionalidades que um outro objeto pode lhe fornecer. Na etapa de projeto do sistema, podemos definir conjuntos de métodos que determinadas classes do nosso sistema deverão implementar incondicionalmente. Tais conjuntos de métodos são as interfaces, as quais contêm a declaração de métodos de forma prototipada, sem qualquer implementação.

Toda classe que implementar uma interface
deverá obrigatoriamente possuir os métodos
predefinidos na interface, caso contrário, resultará
em erro

Prosseguiremos no próximo slide...

Professor: Anderson Henrique

Programador nas Linguagens Java e PHP

