

Introdução a Orientado a Objetos

Analise de Sistemas II

Rogério Fontes
@rogeriofontes

Agenda

- Paradigma Orientação a Objetos
- Conceitos de Orientação a Objetos (OO)
- Classe, Objeto e Mensagem
- Os pilares da Orientação a Objetos (OO)
- Reuso de Implementação

Orientado a Objetos

- A **orientação a objetos** é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Orientação a Objetos (OO)

- É uma forma de **entender** e **representar sistemas complexos** como estruturas hierárquicas de objetos que se **relacionam**.

Paradigma Orientado a Objetos

- É o encurtamento da distância entre a **visão/modelo computacional** e o **mundo real**.
- Na Orientação a Objetos e conversão de **objetos** do **mundo real** para Objetos no computador, baseado no mundo real onde tudo é **composto por objetos**.
- Temos que pensar que objetos são interagindo entre si, criando assim a modelagem para um modelo computacional.

Por que Orientação a Objetos?

- Permite alta reutilização de código;
- Reduz tempo de manutenção de código;
- Reduz complexidade através da melhoria do grau de **abstração** do sistema;

Conceitos da Orientação a Objetos

- Classe
- Objeto (Instância)
- Mensagem
- Encapsulamento
- Herança
- Polimorfismo

Classe

- A classe é a implementação de tipo abstrato de dados no paradigma orientado a objetos.
- Uma classe é um modelo para a criação de objetos. A classe define as propriedades (atributos) e os comportamentos (métodos).
- Além disso, uma classe define como produzir (instanciar) objetos a partir dela.

Classe

- Classe Pessoa – Figura.

Objeto

- Um objeto é uma construção de software que encapsula **estado** e **comportamento**, através respectivamente de **propriedades** (atributos) e **operações** (métodos);
- **Estado de um Objeto**: composto por suas **propriedades** e seus respectivos **valores**;
- **Comportamento**: a maneira como o objeto reage quando o seu estado é **alterado** ou quando uma mensagem é **recebida**.

Objeto

- Cachorro

Mensagens

- Mecanismo através do qual os objetos se **comunicam**, invocando as **operações** desejadas;
- Um objeto (**Emissor**) envia uma mensagem a outro (**Receptor**) que executará uma tarefa.

Os pilares da OO

- Os pilares da OO são mecanismos fundamentais que garantem a filosofia de Orientação a Objetos. São eles:
 - Encapsulamento;
 - Herança;
 - Polimorfismo.

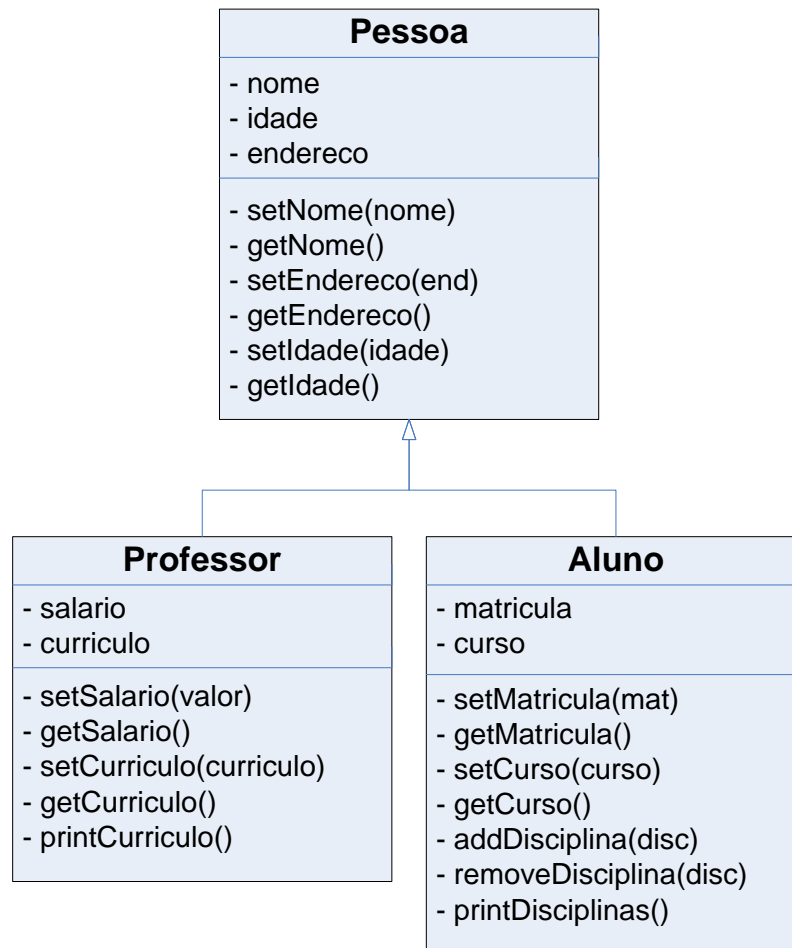
Encapsulamento

- **Resumindo:** “Não mostre as cartas de seu baralho”
- **Objetivos:**
 - Ocultar do mundo externo ao objeto os detalhes de implementação e restringir o acesso às propriedades e aos métodos;
 - Permitir a criação de programas com **menos erros e mais clareza**.
- **Vantagens:**
 - Segurança no acesso ao objeto;
 - Melhor consistência no estado interno, pois evita **alterações incorretas** nos valores das propriedades.

Herança

- Permite especializar **novas classes** (subclasses) a partir de uma classe já existente (superclasse).
- A subclasse herda as **propriedades comuns** da superclasse e pode ainda **adicionar novos métodos** ou **reescrever métodos** herdados.
- **Objetivo:** evitar que classes que possuam atributos ou métodos **semelhantes** sejam **repetidamente criados** (Reutilização de código).

Herança Simples



Polimorfismo

- Permite a um método ter **várias implementações** as quais são selecionadas com base na **quantidade de parâmetros e seus tipos** que é passado para a invocação do método.

```
frear() {  
    Sysout("frear");  
}
```

```
frear(int velocidade) {  
    if (velocidade > 120)  
        Sysout("frear");  
}
```

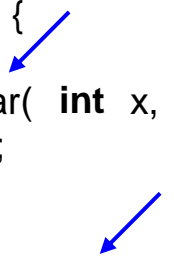
Tipos de Polimorfismo

- São dois os tipos de Polimorfismo:
 - Sobrescrita ou Redefinição de métodos (**Override**);
 - Sobrecarga de métodos (**Overload**).

Exemplo de Sobrecarga (Overload)

- Permite a existência de vários métodos de mesmo **nome**, porém com **assinaturas** levemente diferentes (número, tipo e qtd de parâmetros).

```
public class Soma {  
    public int somar( int x, int y) {  
        return x+y;  
    }  
  
    public double somar( double x, double y) {  
        return x+y;  
    }  
}
```



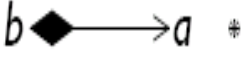
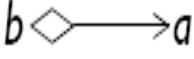

Exemplo de Sobrescrita (Override)

- Permite a existência de vários métodos com **assinaturas idênticas**, porém com implementações distintas.

Reuso de Implementação

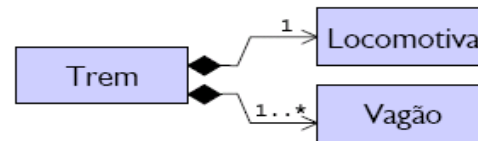
- Separação **interface-implementação** → **maior reuso**
 - Reuso depende de **bom planejamento** durante a etapa de **modelagem**.
- Uma vez criada uma classe, ela deve representar uma **unidade de código útil** para que seja **reutilizável**.

Reuso de Implementação

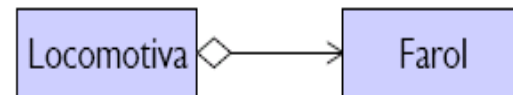
- Formas de uso e reuso de classes
 - Uso e reuso de **objetos** criados pela classe: **mais flexível**
 - **Composição**: a “é parte essencial de” b 
 - **Agregação**: a “é parte de” b 
 - **Associação**: a “é usado por” b 
 - Reuso da interface da classe: **pouco flexível**
 - **Herança**: b “é um tipo de” a (substituição útil, extensão)

Composição, Agregação e Associação

- **Composição** (tem-um): um trem **é formado por** locomotiva e vagões.



- **Agregação**: uma locomotiva **tem** farol (mas não vai deixar de ser uma locomotiva se não o tiver).



- **Associação**: um trem **usa** uma estrada de ferro (não faz parte do trem, mas ele depende dela).

