Classificação, Generalização e Especialização em Orientação a Objetos

Informações do Autor

- **Nome:** Roussian Gaioso
- **Data de atualização**: 02/10/2025

Introdução: Organizando o Mundo Digital

Na vida real, nós organizamos as coisas para entender melhor o mundo. Exemplo: separamos animais em mamíferos, aves, répteis; ou veículos em carros, motos e caminhões.

Na programação orientada a objetos, acontece a mesma coisa: precisamos organizar os elementos do sistema em grupos que façam sentido.

Essa organização torna o software:

- Mais simples de entender (cada coisa no seu lugar).
- Mais fácil de manter (quando algo muda, sabemos onde mexer).
- Mais reaproveitável (uma ideia geral pode servir de base para várias especializações).

Neste guia vamos estudar três ideias centrais:

1. **Classificação** – o ato de agrupar objetos semelhantes em classes.
2. **Generalização e Especialização** – a relação de hierarquia entre classes.
3. **Classes Abstratas e Concretas** – a diferença entre modelos (ideias gerais) e implementações reais.

Classificação: O Primeiro Passo para a Abstração

Na programação orientada a objetos, o primeiro passo é classificar.

Classificar significa observar vários objetos do mundo real, identificar semelhanças e agrupá-los em classes.

Classe é o nome que damos a um grupo de objetos semelhantes, que compartilham características e comportamentos.

Exemplo

Imagine uma lista de diferentes animais e pessoas:

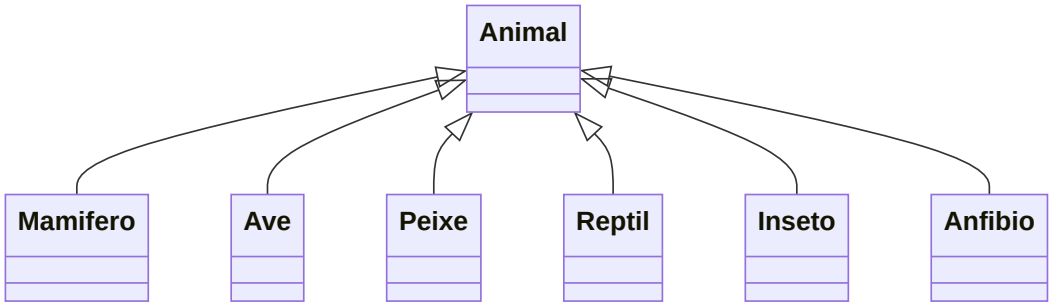
- Elefante, gato, baleia, coelho, homem, mulher, golfinho
- Papagaio, galinha, águia, coruja
- Tubarão, peixe
- Serpente, lagarto
- Abelha, besouro
- Rã

Se olharmos com atenção, conseguimos formar grupos:

- Mamíferos: elefante, gato, baleia, coelho, homem, mulher, golfinho
    - Nascem vivos (não de ovos)
    - São de sangue quente
    - Respiram por pulmões
    - Têm pelos
- Pássaros: papagaio, galinha, águia, coruja
    - Possuem bico
    - Dois pés e duas asas
    - Corpo com penas
    - Põem ovos
- Outros grupos:
    - Peixes: tubarão, peixe
    - Répteis: serpente, lagarto
    - Insetos: abelha, besouro
    - Anfíbios: rã

Resultado da classificação

Agora, em vez de lidar com cada animal separadamente, podemos falar de uma classe inteira, como Mamífero ou Pássaro. Isso reduz a complexidade e permite que pensemos no que é comum a todos os objetos de um grupo.



O próximo passo será organizar essas classes em uma hierarquia, mostrando como elas se relacionam e como podemos reutilizar características entre elas.

Relações Hierárquicas: Estruturando Classes com Superclasses e Subclasses

As classes não vivem sozinhas. Para representar sistemas mais complexos, precisamos organizá-las em hierarquias. Essa estrutura mostra o que é **mais geral** e o que é **mais específico**, ajudando a reduzir a repetição de código e a deixar o sistema mais fácil de entender.

Exemplo: Empresa HomeCare

A empresa HomeCare possui diferentes tipos de funcionários:

- Sean e Sara são vendedores.
- Simon e Sandy são gerentes.

Podemos organizar assim:

- Sean e Sara → objetos da classe Vendedor
- Simon e Sandy → objetos da classe Gerente
- Todos eles → também são objetos da classe Funcionário

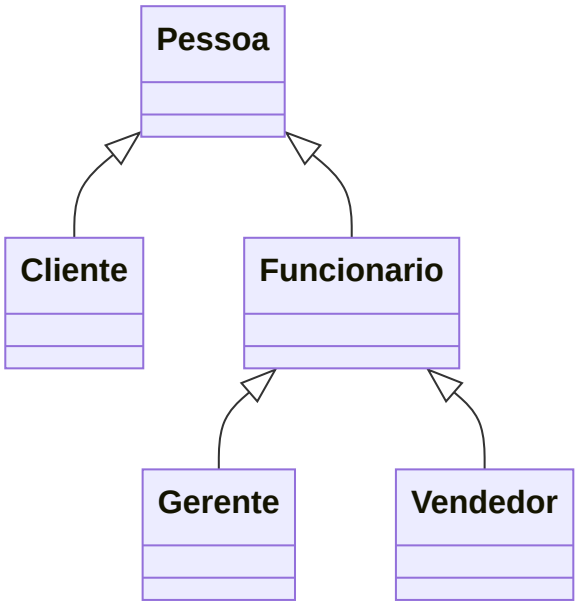
Se chamarmos Sean de Funcionário, estamos olhando de forma **geral** (ele tem salário, férias, registro). Se chamarmos Sean de Vendedor, olhamos de forma **específica** (ele recebe comissão por vendas, algo que não vale para os gerentes).

Termos importantes

- Superclasse → uma classe mais geral, de onde outras herdam.
    - Exemplo: Funcionário é superclasse de Vendedor e Gerente.
- Subclasse → uma classe mais específica, que herda da superclasse.
    - Exemplo: Vendedor e Gerente são subclasses de Funcionário.

Estrutura hierárquica

Podemos visualizar assim:



Note que Funcionário é, ao mesmo tempo:

- Subclasse (de Pessoa)
- Superclasse (de Vendedor e Gerente)

Esse tipo de hierarquia mostra como organizar o sistema de forma clara e lógica.

Na próxima etapa, vamos entender os mecanismos que criam essas hierarquias: generalização e especialização.

Generalização e Especialização: Construindo a Hierarquia

Generalização e especialização são processos complementares.

- Generalização: olhar para o que é comum e criar uma classe mais geral.
- Especialização: olhar para as diferenças e criar classes mais específicas.

Generalização: Abstraindo semelhanças

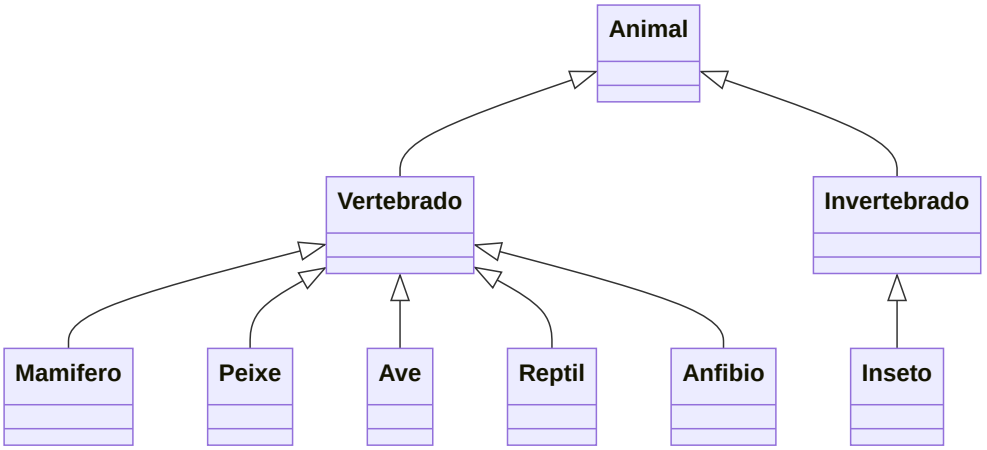
Generalizar significa juntar o que é parecido em uma categoria mais ampla. É um processo de baixo para cima: partimos de classes específicas e subimos para uma classe mais geral.

Exemplo com animais

1. Observamos as classes Mamífero, Peixe, Pássaro, Réptil e Anfíbio.
    - Todas têm espinha dorsal.
    - Podemos criar uma superclasse chamada **Vertebrado**.

2. A classe Inseto não tem espinha dorsal.
    - Ela pode ser generalizada em uma superclasse chamada **Invertebrado**.
3. Tanto Vertebrado quanto Invertebrado são animais.
    - Logo, podemos generalizar em uma classe ainda mais ampla: **Animal**.

Visualização em hierarquia



Esse diagrama mostra claramente a **subida de nível**:

- Vários grupos formam **Vertebrados**.
- Inseto forma os **Invertebrados**.
- Ambos juntos formam **Animal**.

Especialização: Detalhando Diferenças

Especialização é o processo inverso da generalização. Aqui partimos de uma classe mais **geral** e a dividimos em subclasses mais **específicas**, destacando diferenças. É um processo de cima para baixo (top-down).

Exemplo com animais

1. Começamos com a classe geral **Animal**.
    - Podemos especializar com base na presença ou ausência de espinha dorsal.
    - Isso gera as subclasses **Vertebrado** e **Invertebrado**.
2. Dentro de **Vertebrado**, podemos criar subclasses ainda mais específicas:
    - Mamífero
    - Peixe
    - Pássaro

    - Réptil
    - Anfíbio

Assim, cada nível da hierarquia reflete um detalhe a mais sobre os objetos.

A lógica da hierarquia

- Classes mais gerais ficam no **topo** da hierarquia.
- Classes mais específicas ficam na **base**.

Isso significa que:

- Subindo a hierarquia → a definição é mais ampla e vale para muitos objetos.
    - Exemplo: a águia é um **Pássaro**, mas também é um **Vertebrado** e, no nível mais geral, um **Animal**.
- Descendo a hierarquia → a definição é mais detalhada e vale para menos objetos.
    - Exemplo: apenas os **Mamíferos** nascem vivos, não todos os animais.

Essa lógica nos leva a uma pergunta importante: Para que servem essas classes muito gerais, como **Animal**, que nem sempre representam algo concreto? A resposta está na diferença entre **classes abstratas** e **classes concretas**, que veremos a seguir.

Classes Abstratas vs. Classes Concretas: Projetos vs. Objetos Reais

Dentro de uma hierarquia, as classes podem ter propósitos diferentes: algumas existem apenas como **modelos**, enquanto outras são usadas para criar **objetos reais**.

Classe Abstrata

- Uma classe abstrata é tão geral que não faz sentido criar um objeto direto dela.
- Ela serve como **modelo** para as subclasses.
- Define atributos e comportamentos comuns, que serão herdados ou implementados pelas subclasses.

Exemplo com animais:

- Não criamos um objeto "Animal" ou "Vertebrado" sozinho.
- Fazemos instâncias de classes mais específicas, como "Gato" ou "Águia".

```java
abstract class Animal {
    String nome;
    abstract void emitirSom();
}
```

Aqui, `Animal` é abstrata porque não faz sentido instanciar algo tão genérico.

Classe Concreta

- Uma classe concreta pode ser instanciada, ou seja, dela criamos objetos reais.
- Representa elementos específicos do sistema.
- Normalmente está nos níveis mais baixos da hierarquia.

Exemplo com animais:

- Mamífero, Peixe, Pássaro, Réptil, Anfíbio e Inseto são classes concretas.
- A partir delas criamos objetos como:
    - `Mighty` → Elefante (classe Mamífero)
    - `Jaws` → Tubarão (classe Peixe)

```java
class Gato extends Animal {
    @Override
    void emitirSom() {
        System.out.println("Miau");
    }
}

public class Teste {
    public static void main(String[] args) {
        Animal meuGato = new Gato();  // objeto concreto
        meuGato.emitirSom();          // saída: Miau
    }
}
```

Nesse exemplo:

- `Animal` é abstrata → não instanciamos diretamente.
- `Gato` é concreta → dela criamos objetos reais no sistema.

Implementação Prática em Java

O Java possui palavras-chave específicas para diferenciar uma **classe abstrata** de uma **classe concreta**.

1. Definindo uma classe abstrata

Usamos a palavra-chave `abstract`. Ela indica que a classe **não pode ser instanciada diretamente**.

```
abstract class Animal {
    String nome;

    // método abstrato: cada animal vai
    //implementar do seu jeito
    abstract void emitirSom();
}
```

2. Definindo uma classe concreta

As classes concretas são definidas normalmente e herdam das abstratas usando `extends`. Elas devem implementar os métodos abstratos da superclasse.

```
class Gato extends Animal {
    @Override
    void emitirSom() {
        System.out.println("Miau");
    }
}
```

3. Instanciação

- Tentar criar um objeto de `Animal` (abstrata) → gera erro.
- Criar um objeto de `Gato` (concreta) → funciona normalmente.

```
public class Teste {
    public static void main(String[] args) {
        // ERRO: não pode
        // Animal a = new Animal();
        // instanciar abstrata
        // OK: instanciamos a concreta
        Animal meuGato = new Gato();
        // saída: Miau
        meuGato.emitirSom();
    }
}
```

Conclusão

- Uma **classe abstrata** representa um conceito ou modelo.

- Uma **classe concreta** representa a materialização desse conceito em um objeto real.
