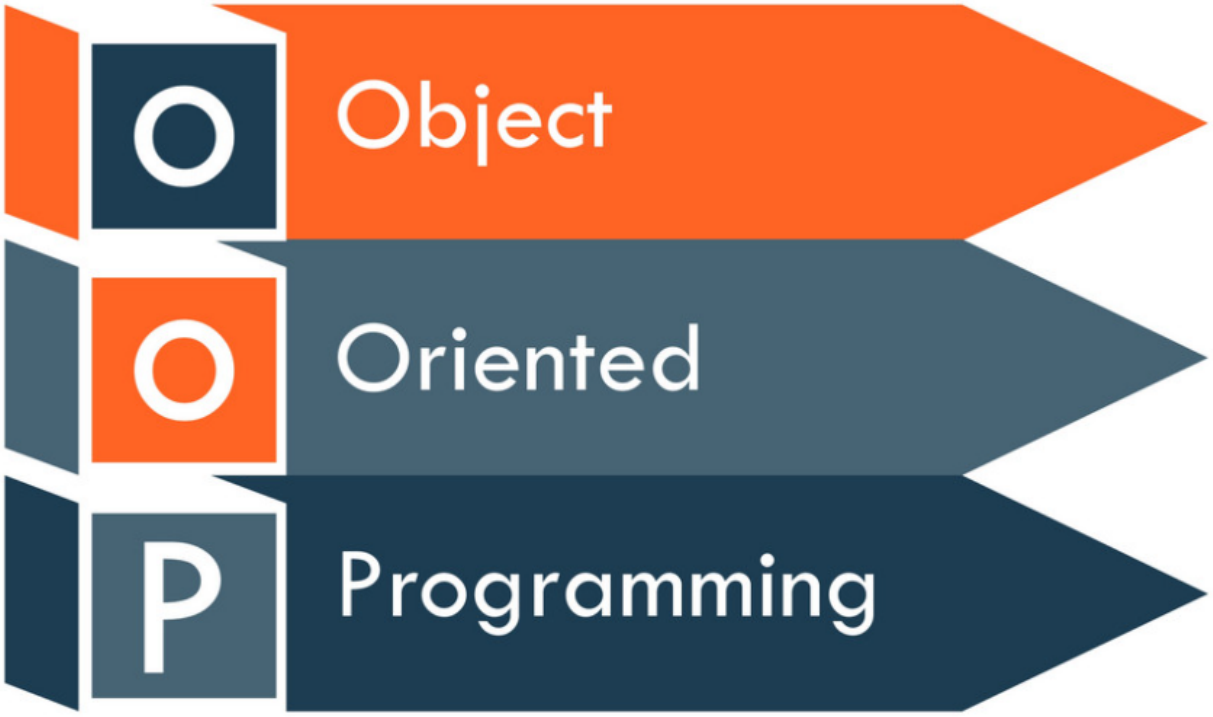
POO - Programação Orientada a Objetos



Diogo Antonio Sperandio Xavier

SUMÁRIO

INTRODUÇÃO

Fundamentos da POO

Conceitos principais
- Objetos
- Atributos e métodos
  - Atributos
  - Métodos
- Classes
  - Instanciação
  - Classes Abstratas ou puras

Princípios básicos da orientação a objetos
- Abstração
- Encapsulamento
- Herança
  - Herança simples
  - Herança Múltipla
- Polimorfismo
- Estruturas
  - Estrutura de Generalização - Especialização
  - Estrutura todo - Parte

Modelagem de um sistema Orientado a Objetos

Programação Orientada a Objetos (POO ou OOP)
- Vantagens e benefícios
  - Unificação entre dados e processos
  - Consistência entre análise e desenvolvimento
  - Reutilização e aumento da produtividade
  - Multidesenvolvimento
  - Facilidades em manutenção

SITES DE APOIO E BIBLIOGRAFIA

INTRODUÇÃO

O objetivo do material em questão é simplificar os conceitos de POO (Programação orientada a objetos) a fim de facilitar o entendimento de quem está começando a programar. **Não são necessários conhecimentos em programação** para compreender os conceitos e a ideia, mas caso possua algum, fica mais fácil o aprofundamento no tema.

Espero que esse livro possa colaborar com seu desenvolvimento de alguma forma, seja com conhecimento ou com um passo-a-passo que melhore sua venda e seus negócios. Caso isso aconteça, já estou de antemão muito feliz por isso.

Fundamentos da POO

Na orientação a objetos, os sistemas são vistos como uma **coleção de objetos que interagem de maneira organizada** com o intuito de melhorar a **reusabilidade** e **a extensibilidade** de um software. Ela tem como fundamento o **modelo de objetos** que engloba a **abstração, hierarquização, encapsulamento, classificação, modularização, relacionamento, simultaneidade e persistência.**

Com esse método, é possível uma representação mais fidedigna do mundo real em sistemas computacionais, já que nós entendemos o mundo como **um composto de vários objetos que interagem entre si**, assim como na orientação a objetos. Uma das grandes vantagens da O.O. está no conceito de **herança**, que permite que definições existentes sejam estendidas. Também deve ser enfatizado o **polimorfismo,** que permite solucionar funcionalidades que um programa irá utilizar dinamicamente durante sua execução.

Conceitos principais

Objetos

Um **objeto** é um elemento computacional que representa, no domínio da solução, alguma **entidade (abstrata ou concreta)** do domínio de interesse do problema sob análise.

Os objetos são instâncias de classe que determinam qual informação um objeto contém e como pode manipulá-la. São capazes de reter informação (estado) e oferecer uma série de comportamentos (operações) para examinar ou afetar o estado.

Ex:

Figura 1 - Cachorro

Analisando um "objeto" cachorro, concluímos que ele possui algumas características como: **Nome, idade, cor do pêlo, cor dos olhos, peso, raça,** entre outras. Essas características se chamam **ATRIBUTOS.**

Além do conjunto de características que o cachorro possui, ele realiza algumas ações como: **Latir, comer, dormir, pegar o graveto, rosnar,** entre outras. As ações que um objeto pode executar são chamadas de **MÉTODOS.**

A única maneira de interagir com os objetos é através dos métodos que ele disponibiliza, como por exemplo, para alimentá-lo utilizamos o método **comer,** ou para brincar o método **pegar o graveto**. Esse conjunto de métodos disponíveis pode ser chamado de **INTERFACE**.

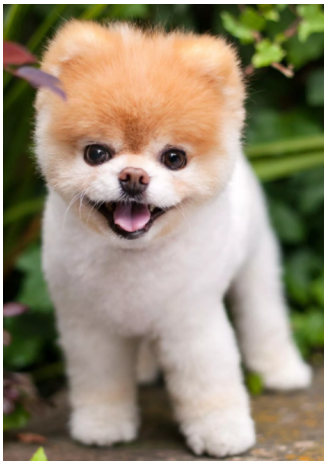
Atributos e métodos

- **Atributos**

Pode-se dizer que esses atributos são campos que armazenam valores ou características do objeto. **O estado de um objeto é o conjunto de valores dos atributos em um determinado instante, já seu comportamento é como ele age e reage** em termos de mudanças de estado e interação com outros objetos.

Seguindo o exemplo, utilizando cachorros, observamos que **o objeto apresenta valores diferentes para os mesmos atributos e esses valores só podem ser mudados através de estímulos** internos ou internos, com eventos que provoquem a transição desses estados em determinado objeto.

Cachorro	
Nome	Gigante
Idade	2 meses
Cor do pelo	Marrom claro
Cor dos olhos	Pretos
Peso	100g
Raça	Pinscher



Cachorro	
Nome	Furiosa
Idade	1 meses
Cor do pelo	Branco e marrom claro
Cor dos olhos	Pretos
Peso	300g
Raça	lulu-da-Pomerânia

- **Métodos**

Os métodos são procedimentos ou funções que realizam as ações próprias do objeto, ou seja, ações que o objeto pode realizar. Tudo que o objeto faz é realizado através de seus métodos. O objeto requisita uma ação de outro objeto enviando uma mensagem para ele. A mensagem é uma solicitação para que o objeto recebedor execute rotinas chamadas **Métodos de classe**, que são responsáveis por acessar ou alterar os atributos de um objeto.

No exemplo dos cachorros os métodos são: **Latir, comer, dormir, pegar o graveto, rosnar...**

Classes

Uma classe **representa um conjunto de objetos que possuem características e comportamentos comuns** e de agora em diante, um objeto será uma instância de uma determinada classe, ou seja, criaremos objetos baseados nas características definidas nas classes.

A ênfase da metodologia de orientação a objetos é dada na criação de classes, ao contrário do que seu nome indica.

Seguindo com o exemplo dos **cães**, é possível observar que ambos possuem o **mesmo conjunto de atributos e métodos. Isso ocorre pois se trata de dois objetos de mesma classe.**

Figura 2 - Classe cães

Outro exemplo seria uma **classe de gatos**. Seus atributos são: **Nome, idade, peso, cor dos pêlos, cor dos olhos, peso e raça.** Seus métodos são: **miar, comer, dormir, subir nos móveis.**

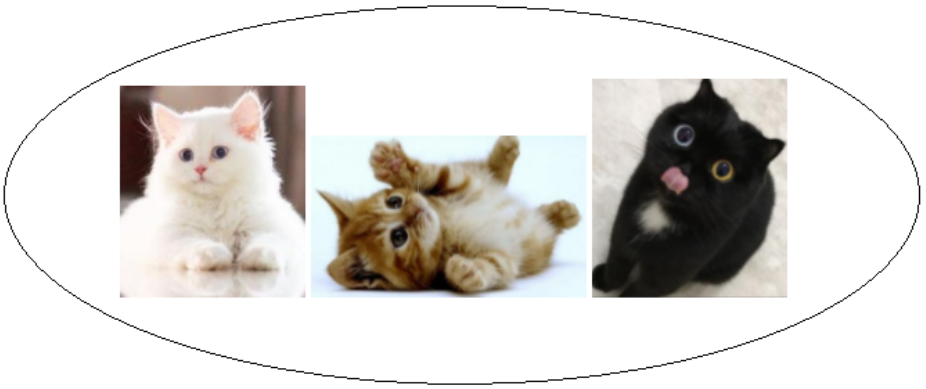


Figura 3 - Classe gatos

Uma classe representa um gabarito para muitos objetos e descreve como estes objetos estão estruturados internamente.

As classes exemplo (cães e gatos) possuem atributos comuns, por exemplo: **Nome, idade, peso, cor dos pêlos, cor dos olhos, peso e raça** e métodos comuns como: **comer e dormir**. Dessa maneira, surge então o conceito de **subclasse** e **superclasse**. Um exemplo de superclasse seriam os mamíferos.

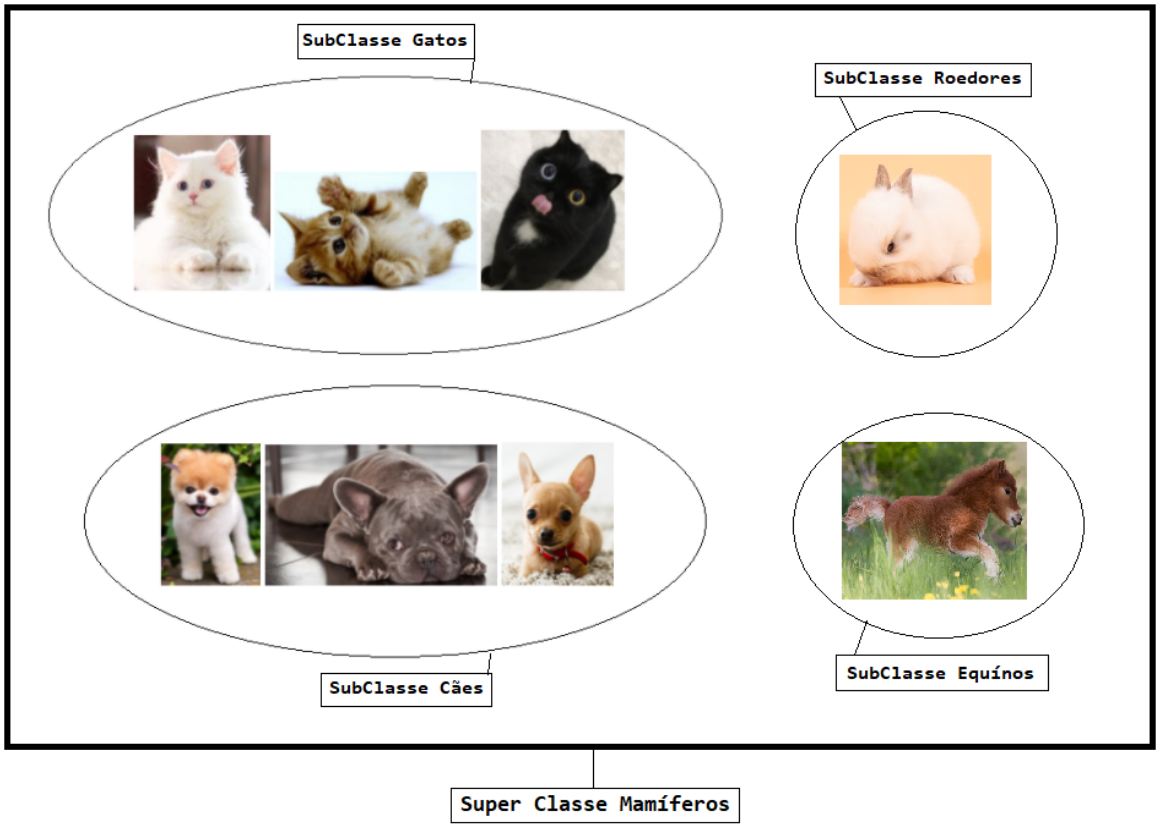


Figura 4 - Superclasse Mamíferos

Dentro da Superclasse Mamíferos temos pelo menos 4 Subclasses. Podemos dizer que elas apresentam os seguintes atributos em comum: **nome, idade, cor dos pêlos, cor dos olhos, peso, raça**. Além disso, estas subclasses possuem os seguintes métodos em comum: **comer e dormir**.

Na figura 4, **temos 8 objetos diferentes distribuídos em 4 subclasses.**

Utilizando a hierarquia de classe, podemos omitir da declaração de um objeto ou de uma classe inferior tudo que foi definido nas classes ou na classe superior. Só serão definidos no objeto os atributos e métodos particulares desse objeto que não são atribuídos aos outros objetos da mesma classe, **ou seja, ao criar um objeto ele terá os atributos da classe, basta definir os que serão omitidos ou adicionados.**

As classes da qual as outras dependem podem ser chamadas de **"ancestrais"** e de **"dependentes"** as classes originadas a partir de outras. No exemplo utilizado, a classe **"Mamíferos"** têm um ancestral **"Animais"** e descendentes **"Cães"**, **"Gatos"** entre outros.

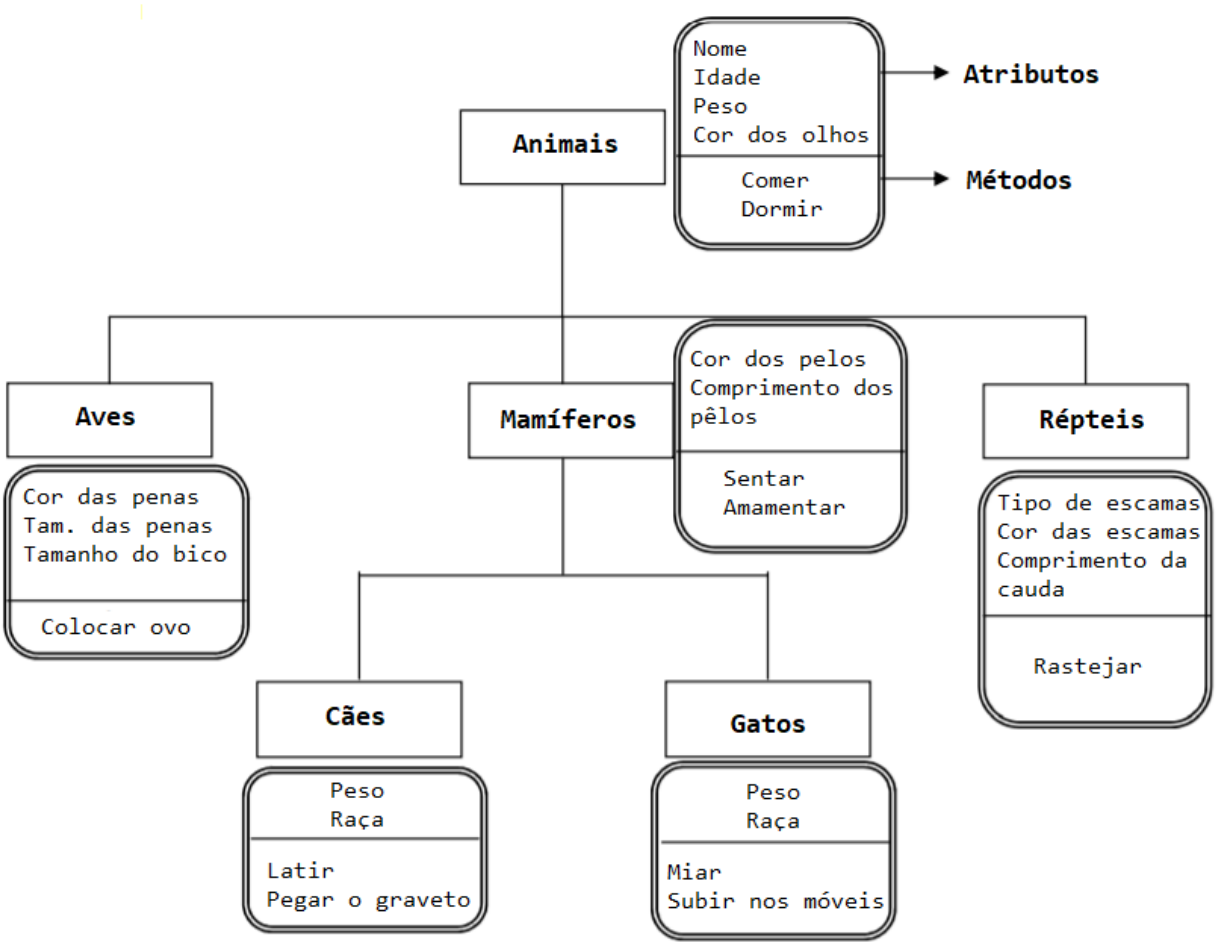


Figura 5 - Hierarquia de classes

As classes são definições de como os objetos devem ser e não existem na realidade. Apenas objetos existem, como por exemplo, as pessoas apresentam os cachorros pelo nome, assim como outras pessoas.

- **Instanciação**

É quando a classe produz um objeto como um modelo, ou gabarito, para criação de objetos. Sendo assim, **um objeto nada mais é do que a instância de uma classe**.
Pelo exemplo que foi feito acima, cada cachorro que for armazenado é um novo objeto, uma nova instância da classe "Cães". **A classe serve de modelo para a criação de novos objetos**.

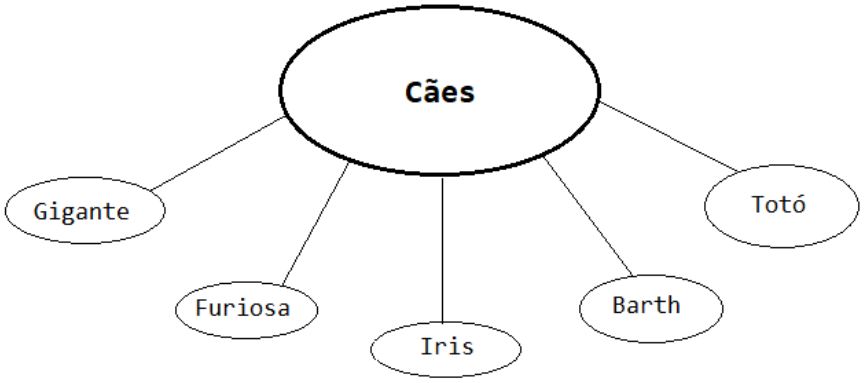


Figura 6 - Instanciação

- **Classes Abstratas ou puras**

Classes abstratas (ou puras) **são classes das quais os objetos nunca são instanciados diretamente, mas sempre por uma classe descendente dela**. São criadas para facilitar o processo de estruturação. Um exemplo clássico é **criar uma classe Pessoa,** com atributos (nome, endereço, telefone, etc.) e métodos (alterar endereço, alterar telefone, imprimir cadastro, etc.) necessários para manusear dados de pessoas em um sistema de informação. **A partir dessa classe genérica se cria classes descendentes específicas para manusear como Funcionário e Gerente.**

Figura 7 - Exemplo de classe abstrata

A classe **Pessoa** nunca terá um objeto a ela instanciado, ela só existe para unificar todos os atributos e métodos comuns às classes **Gerente** e **Funcionário** evitando assim a redundância.

Princípios básicos da orientação a objetos

Abstração

O significado da palavra é "ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora da totalidade". **A Abstração é o princípio pelo qual cada componente deve manter oculta, sob sua guarda, uma decisão de projeto única**. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado.

O recurso da abstração é utilizado para entender problemas complexos, dividindo assim esse problema em problemas menores. Após isso, resolvemos cada um deles até encontrarmos a solução do problema inteiro. Por esse princípio, isolamos os objetos que queremos representar do ambiente complexo em que se situam, e nesses objetos, representamos só as características que são relevantes para o problema em questão. O uso da **Abstração** permite também que determinadas partes do problema ganhem maior ou menor peso, e dessa maneira, detalhes podem ser desconsiderados em determinados momentos para que outros sejam ressaltados.

Encapsulamento

É o princípio de projeto pela qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação como uma unidade (cápsula). Aliado ao conceito de abstração, podemos ocultar detalhes de uma estrutura complexa, que poderiam interferir na análise. Na **orientação a objetos** os dados e processos que tratam determinados dados estão **encapsulados** numa única entidade. **A única maneira de conhecer ou alterar os atributos de um objeto é através de seus métodos**.

O **encapsulamento** é um dos grandes trunfos em relação à programação estruturada (tradicional). **A vantagem** é que **o encapsulamento disponibiliza o objeto com toda sua funcionalidade sem que precise saber como ele funciona internamente** nem **como armazena os dados que serão recuperados**.Ex: Ligação telefônica, onde apenas digitamos o número e acontece a chamada sem termos acesso aos processos por trás.

Outro ponto importante é o fato de **permitir modificações internas em um objeto, adicionando métodos sem afetar os outros componentes do sistema que utilizam o mesmo objeto**. No exemplo da linha, a companhia pode aumentar a quantidade de serviços oferecidos sem modificar o método que é usado para telefonar.

Herança

É aquilo que se herda, que se transmite por hereditariedade. Na programação orientada a objetos, **Herança** é o mecanismo pela qual uma classe obtém as características e métodos de outra para expandi-la ou especializá-la de alguma forma, ou seja, **uma classe pode "herdar" características e atributos de outras classes. Da mesma forma, uma classe transmite suas características para outras, tornando as que receberam suas herdeiras.**

No ponto de vista prático, **a herança é um mecanismo inteligente de aproveitar código** e, através dela, **os objetos podem compartilhar métodos e atributos**. Com isso cria-se classes "filhas" com métodos e atributos herdados.

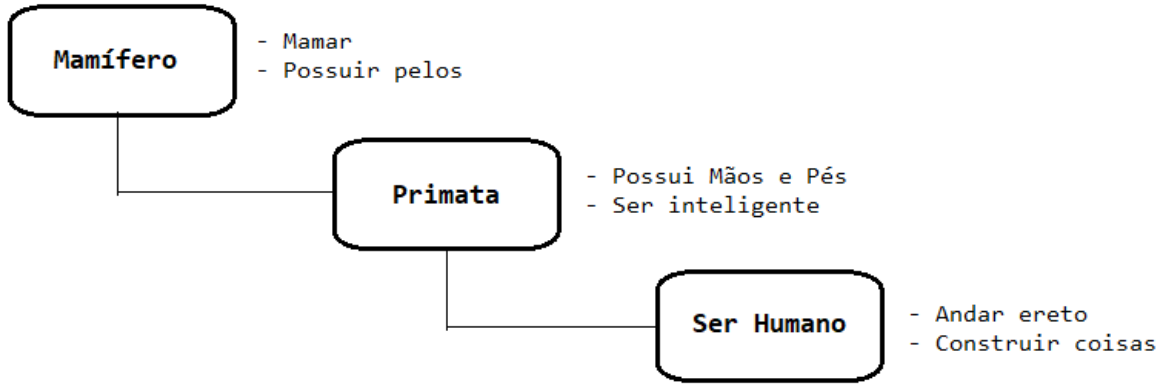


Figura 8 - Exemplo de herança

Na figura 8, temos a classe **Primata** como **subclasse** de **Mamífero** e como **superclasse** de **Ser Humano**. Os métodos acima são passados para seus herdeiros, dessa forma, **Andar ereto** e **Construir coisas** são exclusivos da classe **Ser Humano**, enquanto **Ser inteligente,** por exemplo, é um método da classe **Primata**, que é herdado pela classe **Ser Humano**. A lógica vale para todos os outros métodos.

Reforçando: **Herança significa que todos os atributos e métodos programados no ancestral já estarão automaticamente presentes em seus descendentes sem a necessidade de serem reescritos.**

Além da **economia de código**, a herança garante a **integridade do código**, pois quando alteramos um comportamento em uma classe, as classes descendentes também utilizaram dessa mudança sem a necessidade de reprogramação.

- **Herança simples**

Uma herança é denominada simples quando uma classe herda características **de apenas UMA Superclasse.** Ex: **classe Pessoa** que deriva uma **Classe Funcionário**. A classe **Funcionário** herda todas as características da Superclasse **Pessoa** e de mais nenhuma outra. A **classe Funcionário** possui o diferencial de ter métodos e atributos que a **Pessoa** não possui, como salário e função, por exemplo. A **classe Pessoa** pode gerar uma **classe Cliente** e outras mais, pois o conceito de **herança simples** se refere a **origem das características herdadas pertencerem a apenas uma superclasse**.

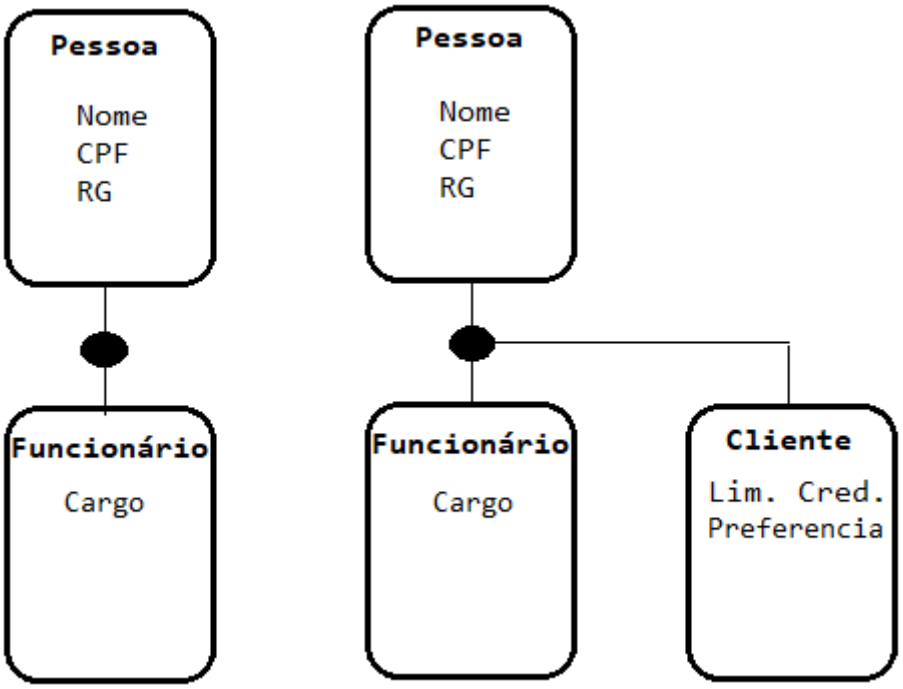


Figura 9 - Ex. Herança simples com um e com dois descendentes

- **Herança Múltipla**

Uma herança é denominada múltipla quando herda características de **DUAS OU MAIS superclasses**. Utilizando o mesmo exemplo acima, funcionário e cliente, temos o caso que o **funcionário pode ser um cliente**, e dessa forma, **além de possuir os atributos que o qualificam como funcionário, também deve possuir os atributos e métodos que o qualificam como cliente**.

Nesse caso, cria-se uma nova subclasse chamada **Funcionario-Cliente** que tem como superclasse as **classes Funcionário** e **Cliente**. Essa nova classe

herda todas as características da **superclasse Pessoa, Funcionário e Cliente.** O seu diferencial pode ser exatamente o fato de possuir atributos e métodos da **classe Funcionário** e **classe Cliente** juntas, mas nada impede que tenha atributos próprios como um desconto especial.

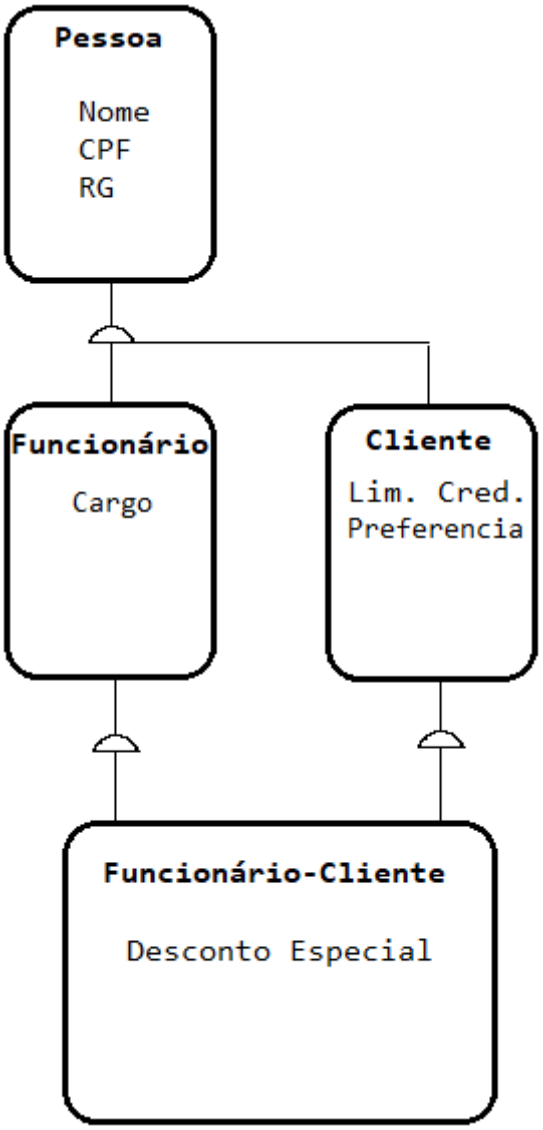


Figura 10 - Ex. Herança múltipla

Polimorfismo

É a capacidade de uma variável se transferir em tempo de execução a objetos de diversas classes. Também pode ser definido como a capacidade que objetos diferentes têm de responder a uma mesma mensagem. Uma mesma mensagem pode apresentar formas de execução diferentes, próprias de cada objeto. **Com esta característica o usuário pode enviar uma mensagem genérica e abandonar detalhes sobre a exata implementação sobre o objeto receptor.**

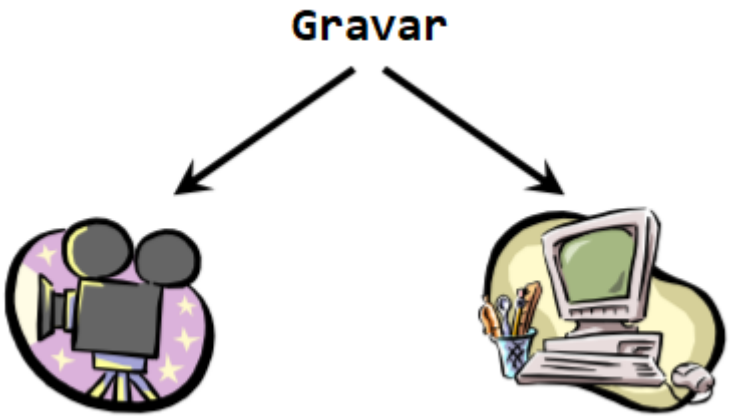


Figura 11 - Ex. Polimorfismo

Ambos objetos (filmadora e computador) possuem o método **“gravar”**, porém ele é executado de forma diferente. O **polimorfismo está diretamente ligado à hereditariedade de classes. O polimorfismo ocorre quando um método já definido em seu ancestral é redefinido no descendente com um comportamento diferente.**

Estruturas

As estruturas ajudam os analistas a arranjar os objetos de forma que seja possível visualizar melhor o domínio e a complexidade do problema em estudo. Na análise orientada a objetos existem dois tipos básicos de estrutura: **Generalização-Especialização** e **Todo-Parte.**

- **Estrutura de Generalização - Especialização**

Utilizando o exemplo da herança simples temos a **classe Pessoa** com a mais alta **classe generalizada,** sendo então a **superclasse de estrutura.** Logo abaixo encontramos as duas **subclasses, Funcionário** e **Cliente,** que são especializações da **classe Pessoa,** ou podemos enxergar a **classe pessoa** como generalização da **classe Funcionario** e da **classe Cliente.**

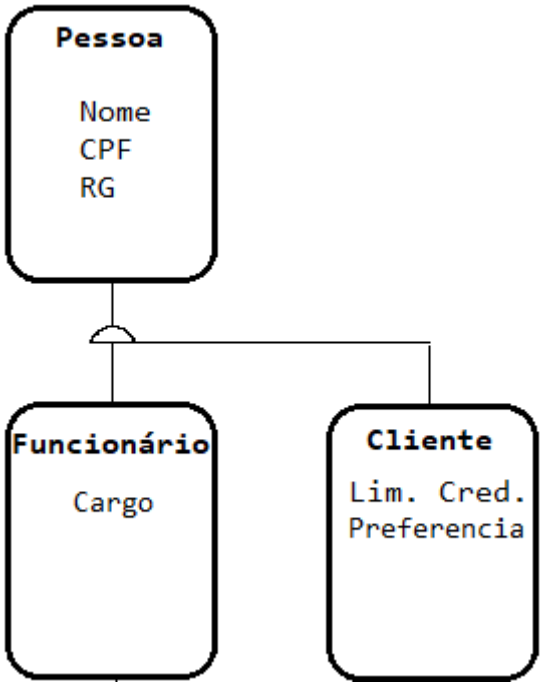


Figura 12 - Ex. estrutura de Generalização-Especialização

A estrutura é formada por uma **classe genérica** no topo e suas **classes descendentes especializadas** abaixo. O uso do semicírculo identifica a classe genérica e as especializações e a linha saindo do ponto central dele aponta para a generalização.

Essa estrutura é, basicamente, uma estrutura hierárquica onde temos superclasses e suas respectivas subclasses. A estrutura de Generalização-Especialização é, de fato, o processo de herança discutido anteriormente. Dessa forma, desejando-se reutilizar uma herança, deve-se utilizar esta estrutura.

- **Estrutura todo - Parte**

Este tipo de estrutura é bastante característico. **Se trata de agregação ou decomposição de objetos**. Essa estratégia é bastante útil na identificação dos objetos e dos seus componentes diante de um determinado problema. Nessa estrutura descreve-se, quando aplicável, quais **subclasses** formam **superclasses. Em outras palavras, um objeto ou uma classe podem conter dentro de si outras estruturas. Com isso, nosso entendimento do problema se reduz ao estado do problema em questão, tornando a análise mais clara e objetiva.**

A notação gráfica é um triângulo que aponta para as **superclasses**, e dessa forma, sabemos quem é composto por quem. No exemplo abaixo, vamos observar um objeto **Carro** que pode ser decomposto em diversos outros objetos, como motor, rodas, banco e etc. Essa divisão pode se estender para os componentes citados, tornando possível perceber que **um objeto complexo nada mais é que um agregado de objetos simples.**

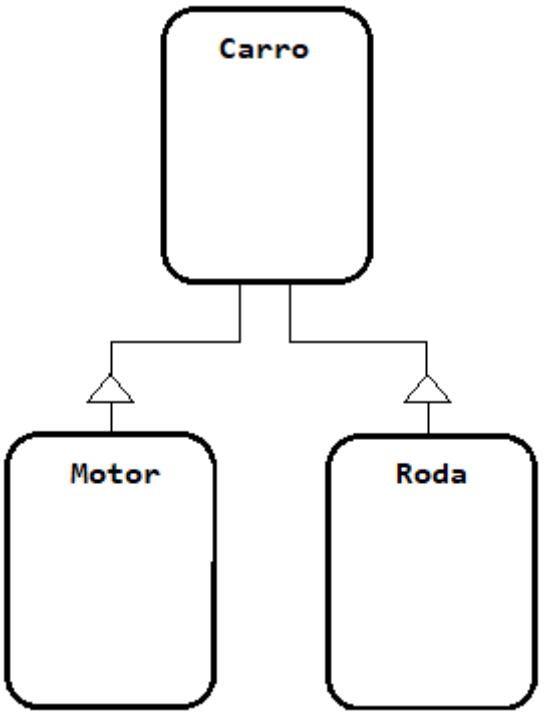


Figura 13 - Ex. estrutura Todo-Parte

A estrutura Todo-Parte possui também um componente chamado **cardinalidade.** Esse componente representa o **quanto possui**. Considerando apenas o motor de combustão do carro e que as rodas são iguais, podendo ou não possui estepe, a **cardinalidade do motor com o carro é 0,1** e do **carro com o motor 1,1**, no caso da roda, a **cardinalidade da roda com o carro é 0,1** e do **carro com a roda 4,5.**

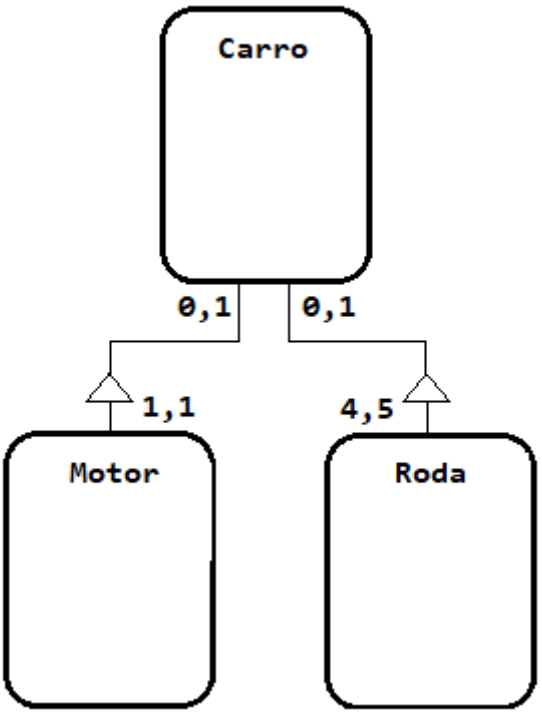


Figura 14 - Cardinalidade

Modelagem de um sistema Orientado a Objetos

A modelagem baseada em objetos é **um meio para descrever os sistemas do mundo real**. Um objeto representa uma abstração de uma entidade do mundo real combinando, em um mesmo elemento, abstração e comportamento.

É possível modelar uma solução pensando totalmente orientado a objetos desde a fase de análise, passando pelo projeto do software e chegando à implementação em uma linguagem de programação orientada a objetos. **Uma grande vantagem de se pensar totalmente orientado a objeto é o fato de que um mesmo objeto, concebido na análise, passa com as mesmas características desde o usuário até o programador que o codifica.**

Apesar de muitas técnicas de modelagem de sistemas com base em objetos e filosofias difundidas, grande parte dos autores converge quando se trata do ponto que **um processo de análise se inicia com a identificação das classes dos objetos. A ideia é que, uma vez com as classes dos objetos definidas, o analista, expandindo e ajustando essas classes e objetos também acabe por melhorar o modelo.** Esse processo é natural e esperado.

Normalmente os processos básicos a serem observados para o desenvolvimento de um modelo orientado a objetos são:

1. **Identificar as Classes e Objetos;**
2. **Identificar as Estruturas e os Relacionamentos entre os Objetos;**
3. **Identificar os Atributos e Métodos importantes.**

O **foco principal da análise orientada a objetos** não está na estrutura ou nos dados do sistema, mas sim **na composição das classes e objetos do sistema**. É muito importante que tanto as **classes quanto os objetos estejam bem definidos** antes de começar qualquer linha de código, pois estão amarrados de tal modo que mesmo com uma alteração pequena pode acarretar uma mudança que tende a se propagar por todo o sistema modelado.

Programação Orientada a Objetos (POO ou OOP)

A programação orientada a objetos consiste em utilizar estruturas de dados que simulem o comportamento dos objetos, facilitando a programação pelo uso de uma metodologia unificada para análise e a programação. Os mesmos objetos identificados pelo analista no diagrama podem ser implementados exatamente como foram projetados, sem necessidade de fazer uma "tradução" dos diagramas de análise para estruturas de dados e de programação como acontece na análise estruturada. **Basicamente, a POO consiste em utilizar objetos computacionais para implementar as funcionalidades de um sistema.**

Nota do pessoal: Eu, Diogo, considero o conceito mais simples de entender e de visualizar, é uma forma que **se exercitada** se torna muito intuitiva de observar os problemas à nossa volta.

Vale destacar que é possível fazer uma análise orientada a objetos e implementar em uma linguagem tradicional, bem como é possível fazer implementações em linguagens orientadas a objetos de sistemas analisados com outras metodologias.

As técnicas de programação orientadas a objetos recomendam que **a estrutura do objeto e a implementação de seus métodos devem ser tão privativas quanto possível**. Normalmente os atributos de um objeto não devem ser visíveis externamente da mesma forma que um método deve ser suficiente para conhecer apenas sua especificação, sem necessidade de saber detalhes como a funcionalidade que ele executa.

Vantagens e benefícios

A Orientação a Objetos consiste em conceber um sistema computacional como um todo orgânico formado por objetos que se relacionam entre si. Esse enfoque pode ser aplicado tanto à análise de sistemas quanto à programação, o que vem a ser uma das principais vantagens: **Utilização da mesma metodologia tanto para definição lógica do sistema quanto para a sua implementação.**

- **Unificação entre dados e processos**

Na orientação a objetos, **os dados e processos são apenas componentes, e o enfoque está em identificar quais os objetos que interagem entre si no sistema**. Os dados são identificados procurando os atributos que definem objetos, e os procedimentos pelas operações que estes objetos realizam. A interação entre os objetos é definida pelas estruturas e relacionamentos

que são identificados. **O resultado é que em um modelo orientado a objetos, existe total coerência entre os dados e os processos, mesmo quando há muitas pessoas trabalhando no mesmo sistema.**

- **Consistência entre análise e desenvolvimento**

Tanto na análise quanto no desenvolvimento, as classes e objetos definidos na análise são exatamente os mesmos que serão implementados. Dependendo do ambiente e da linguagem escolhidos, será necessário acrescentar objetos a mais na implementação, normalmente para o controle da interface, reduzindo o número de problemas oriundos de erro de tradução entre análise e implementação.

- **Reutilização e aumento da produtividade**

A reutilização na orientação a objetos é muito mais do que copiar funções ou utilizar bibliotecas de funções. A reutilização de objetos é de fato utilizar objetos já utilizados em sistemas anteriores em novos sistemas sem modificar suas estruturas internas ou mesmo sem a necessidade de fazer modificações no sistema para acomodar novos códigos.

Com objetos já encapsulados, a reutilização pode somar força quando vamos projetar um novo sistema de informação. Podemos também transformar objetos que possuam características semelhantes aos objetos necessários em objetos úteis ao novo sistema com pequenas modificações. Uma terceira forma de reutilizar objetos é utilizando a propriedade da herança entre objetos, ou seja, caso surja necessidade de um novo objeto cuja ideia esteja baseada em um objeto já existente, podemos criar novos objetos herdando características dos objetos já desenvolvidos.

- **Multidesenvolvimento**

No contexto da orientação a objetos, o desenvolvimento é uma realidade facilitada devido ao fato de que o funcionamento interno dos objetos é irrelevante para sua utilização, ou seja, **qualquer desenvolvedor pode utilizar objetos desenvolvidos por qualquer outro, bastando para isso conhecer os métodos necessários para utilizá-lo.**

- **Facilidades em manutenção**

Falando em manutenção e sistemas orientados a objetos, falamos em manutenção de objetos. Novamente a atenção está concentrada no objeto e não em um sistema com suas interligações. Assim, todas as modificações necessárias para fazer funcionar as novas tarefas do sistema em manutenção ficam restritas às modificações de comportamento dos objetos

envolvidos no contexto da nova necessidade. Com essas alterações sendo efetuadas pontualmente dentro do código do objeto que implementa o comportamento a ser alterado, não existem dúvidas quanto ao local da alteração. É só localizar o programa onde o objeto em questão está sendo implementado.

SITES DE APOIO E BIBLIOGRAFIA

1. https://developer.mozilla.org/pt-BR/docs/Glossary/OOP
2. https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285

- COAD, Peter e YOURDON, Edward. **Análise Baseada em Objetos**. Rio de Janeiro Editora Campus, 1992.

- TAFNER, Malcon A. e CORREA, Carlos H. **Manual de Análise Orientada a Objetos**.