Programação Orientada a Objetos

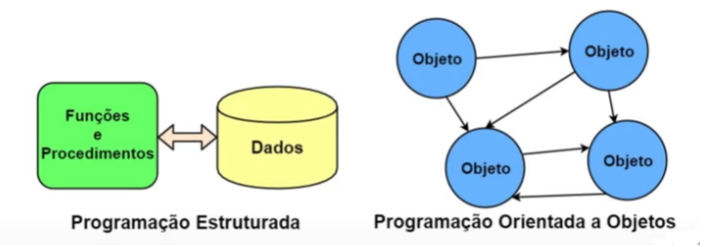
Porque devemos utilizar?

Paradigma Estruturado tem uma representação mais simplista (linguagem C), foca em operações(funções) e dados, ou seja, foca mais no “como fazer”.

Paradigma Orientado a Objeto tem uma representação mais realista, foca na modelagem de entidades e nas interações entre estas, ou seja, foca mais no “o que fazer”.

Vantagens POO

* Melhor coesão (idéias que trabalham juntos);
* Melhor acoplamento, maior flexibilidade;
* Redução do Gap semântico (diferença entre o mundo computacional e real) ;
* Coletor de lixo, a medida que o software vai trabalhando ele vai limpando e liberando a memória conforme tenha coisas que não se utilize mais.



Fundamentos

O que é a Orientação a Objeto

“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.” – Foca, no que queremos resolver, devemos entender bem e analisar o que vamos resolver.

* Abstração - Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos. Preocupa-se com o essencial.
* Reuso - Capacidade de criar novas unidades de código a partir de outras já existentes.
* Encapsulamento – Capacidade de esconder complexidades e proteger dados.

Estruturas Básicas da OO

* Classe: É uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos. Em outros termos, uma classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar, é a base de tudo. Exemplos: Bola, Venda, Carro, Viagem, enfim, entidades do mundo real. Como criar? Substantivos, nomes significativos e o contexto deve ser considerado.
* Atributo: É o elemento de uma classe responsável por definir sua estrutura de dados. O conjunto destes será responsável por representar suas características e fará parte dos objetos criados a partir da classe. Exemplos: Bola -> diâmetro, Carro -> cor, Venda -> valor, características dos objetos. Como criar? São substantivos e adjetivos, nomes significativos, contexto deve ser considerado, abstração e tipos adequados.
* Atributo x Variável

A: O que é próprio e peculiar a alguém ou a alguma coisa.

V: Sujeito a variações ou mudanças; que pode variar; inconstante, instável

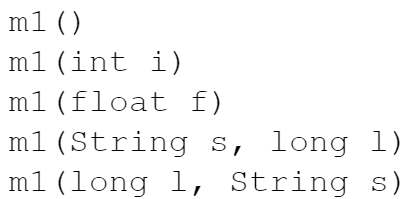
* Métodos: É uma porção de código (sub-rotina) que é disponibilizada pela classe. Este é executado quando é feita uma requisição a ele. Um método serve para identificar quais serviços, ações, que a classe oferece. Eles são responsáveis por definir e realizar um determinado comportamento. Exemplos: Carro -> Ligar, Venda -> Calcular, Computado -> desligar. Como criar? Visibilidade, retorno, nome e parâmetros.
* Dois métodos especiais:

- Construtor;

- Destrutor;

* Sobrecarga: Muda a assinatura de acordo com a necessidade.

- Assinatura: nome + parâmetros.



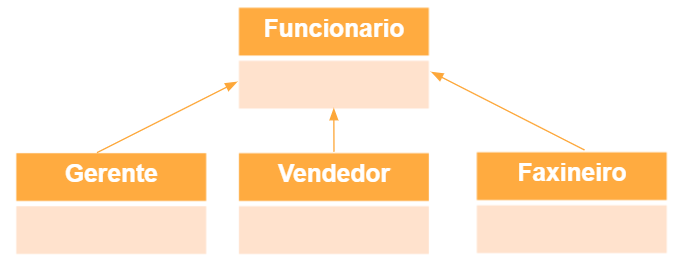
* Um objeto é a representação de um conceito/entidade do mundo real, que pode ser física (bola, carro, árvore etc.) ou conceitual (viagem, estoque, compra etc.) e possui um significado bem definido para um determinado software. Para esse conceito/entidade, deve ser definida inicialmente uma classe a partir da qual posteriormente serão instanciados objetos distintos.
* Mensagem: É o processo de ativação de um método de um objeto. Isto ocorre quando uma requisição (chamada) a esse método é realizada, assim disparando a execução de seu comportamento descrito por sua classe. Pode também ser direcionada diretamente à classe, caso a requisição seja a um método estático.

Herança

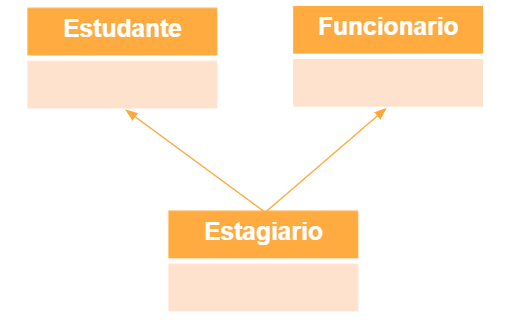
É o relacionamento entre classes em que uma classe chamada de subclasse (classe filha, classe derivada) é uma extensão, um subtipo, de outra classe chamada de superclasse (classe pai, classe mãe, classe base). Devido a isto, a subclasse consegue reaproveitar os atributos e métodos dela. Além dos que venham a ser herdados, a subclasse pode definir seus próprios membros.

Tipos de herança:

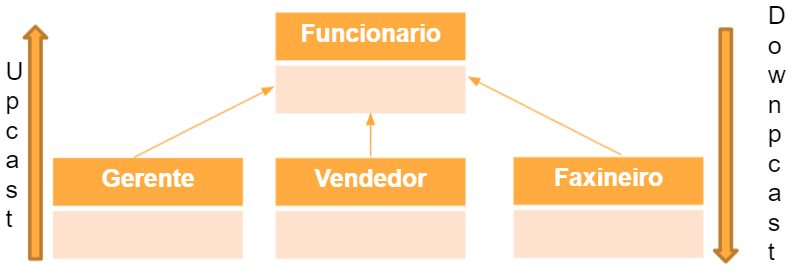
* Simples: A classe filha tem só uma classe mãe.



* Múltipla: A classe filha tem uma ou mais classes mães.

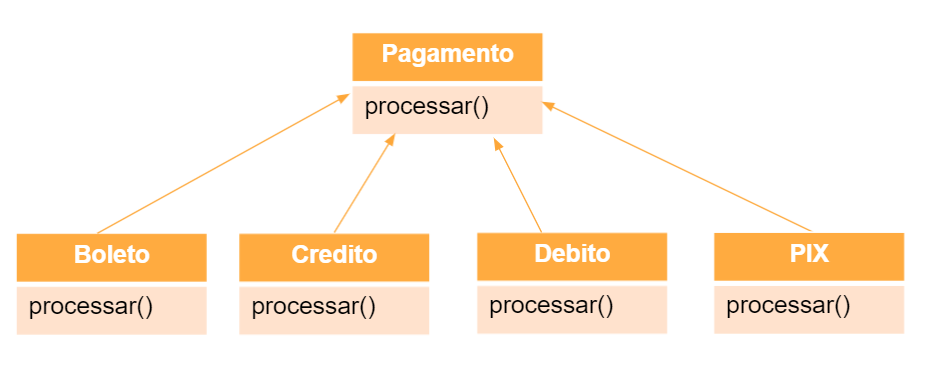


* Upcast e Downcast:

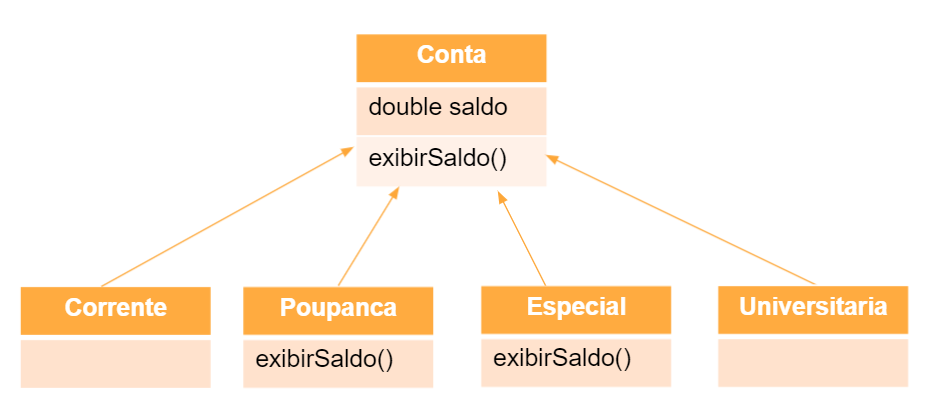


Polimorfismo

A mesma ação, se comportando diferente.



* Sobrescrita

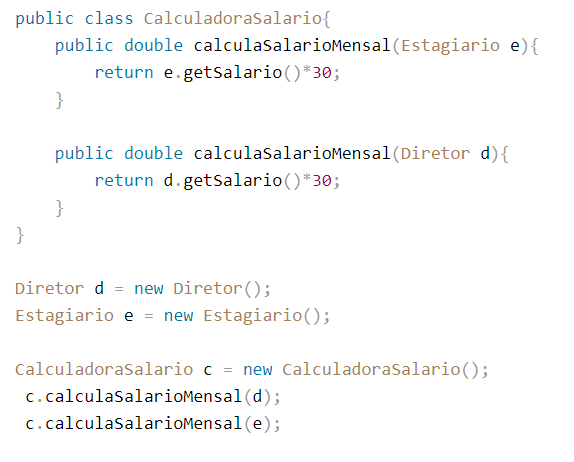


Polimorfismo x Sobrescrita

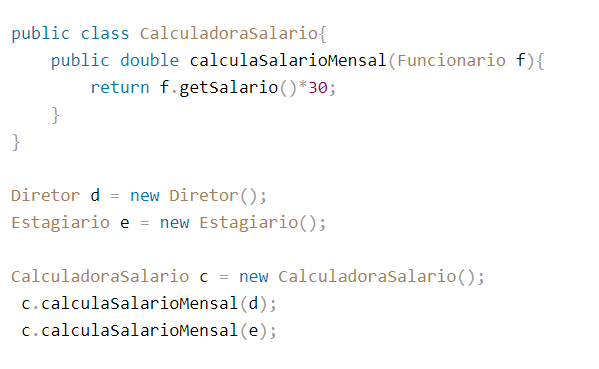
* Polimorfismo -> seria poder referenciar um objeto filho pela sua classe pai.

Exemplo: Vamos usar as classes "Funcionário", "Diretor" e "Estagiário" e vamos supor que o método getSalario() retorna o salário de 1 dia trabalhado, ok?

Agora alguem da minha empresa pediu pra eu criar uma classe que calcule o salário mensal de cada funcionário. Ok... Fácil!



Percebeu que pra cada classe eu tive que criar um método que calcula o salário? uma para o diretor, outra pra o estagiário. Isso parece errado! Se eu tivesse 10 classes para cada profissao eu teria que criar 10 métodos diferentes? Nesse caso uma boa aplicação é o uso do polimorfismo.



Como:

Diretor É UM FUNCIONÁRIO porque ele herda de funcionário.

Estagiário É UM FUNCIONÁRIO porque ele herda de funcionário.

Eu posso criar um método que recebe uma classe pai genérica "Funcionário"

* Sobrescrita -> é quando eu tenho uma classe pai que possui um método e minha classe filha sobrescreve o método do pai.

Exemplo: 

Conforme o exemplo, eu tenho minha classe Funcionário comum na qual o método getSalario retorna as horas trabalhadas vezes 100 reais. Já na classe Estagiário o mesmo método foi sobrescrito (herdado da classe Funcionário) e retorna as horas vezes 50 reais. E na classe diretor o método foi sobrescrito (herdado da classe Funcionário) e retornam as horas vezes 150 reais.

Associação

Possibilita um relacionamento entre classes/objetos, no qual estes possam pedir ajuda a outras classes/objetos e representar de forma completa o conceito ao qual se destinam. Neste tipo de relacionamento, as classes e os objetos interagem entre si para atingir seus objetivos.

A associação entre dois objetos ocorre quando eles são completamente independentes entre si, mas eventualmente estão relacionados. Ela pode ser considerada uma relação de muitos para muitos. Não há propriedade (ownership) nem dependência entre eles. A relação é eventual.

Um exemplo é a relação entre um professor e alunos. Um aluno pode ter vários professores e um professor pode ter vários alunos. Um não depende do outro para existir. Professores podem existir sem alunos e e alunos podem existir sem professores (pelo menos em requisitos normais).

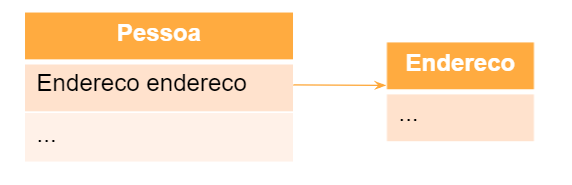
Tipos:

* Estrutural

- Composição: "Com Parte Toda” -> é uma agregação que possui dependência entre os objetos, ou seja, se o objeto principal for destruído, os objetos que o compõe não podem existir mais. Há a chamada relação de morte.

Um exemplo é a relação entre uma a universidade e os departamentos. Além da universidade possuir vários departamentos, eles só podem existir se a universidade existir. Há uma dependência.

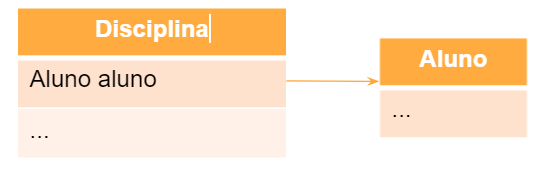
Ex: Pessoa e Endereço



- Agregação: "Sem Parte Toda” -> A agregação não deixa de ser uma associação mas existe uma exclusividade e determinados objetos só podem se relacionar a um objeto específico. É uma relação de um para muitos. Um objeto é proprietário de outros mas não há dependência, então ambos podem existir mesmo que a relação não se estabeleça. Na verdade há controvérsias sobre a exata definição e o que é mais importante, a relação de um para muitos ou a propriedade.

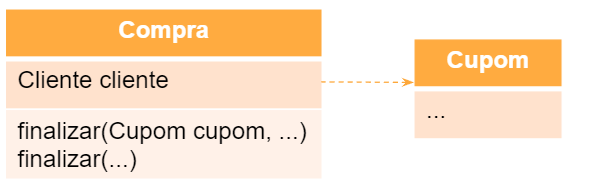
Um exemplo é a relação entre os professores e os departamentos. Departamentos podem ter vários professores. E o professor só pode estar vinculado a um único departamento. Mas eles são independentes. Um professor pode existir sem vínculo com um departamento e este não depende de professores para existir.

Ex: Disciplina e Aluno



* Comportamental

- Dependência: "Depende de" -> Uma mudança em uma classe afeta a mudança em sua classe dependente. Exemplo - o círculo depende da forma (interface). Se você alterar a Forma, isso afetará o círculo também. Portanto, o círculo depende da forma.



Interface

Define um contrato que deve ser seguido pela classe que a implementa. Quando uma classe implementa uma interface, ela se compromete a realizar todos os comportamentos que a interface disponibiliza.

Exemplo: Impressoras e computadores, a impressora HP funciona em varias interfaces diferente, Windows, Linux, Mac, enfim.

Pacotes

São uma organização física ou lógica criada para separar classes com responsabilidades distintas. Com isso, espera-se que a aplicação fique mais organizada e seja possível separar classes de finalidades e representatividades diferentes.

Visibilidade

Um modificador de acesso tem como finalidade determinar até que ponto uma classe, atributo ou método pode ser usado. A utilização de modificadores de acesso é fundamental para o uso efetivo da Orientação a Objetos. Algumas boas práticas e conceitos só são atingidos com o uso correto deles.

* Tipos:
  + Private – Só dentro a classe;
  + Protected – Dentro da classe, mesmo pacote e subclasses;
  + Public – Em qualquer lugar.