



Programação Orientada a Objetos

Educação Profissional

Direitos desta edição

Fundação Bradesco

Homepage

<https://fundacao.bradesco/>

Autoria

Departamento de Educação Profissional e Educação de Jovens e Adultos

Revisão técnica pedagógica

Departamento de Educação Profissional e Educação de Jovens e Adultos

Cintia Batista Pinto da Silva
Luis Ricardo De Oliveira
Marcelo Silva Reis

Coordenação

Departamento de Educação Profissional e Educação de Jovens e Adultos

Rosa Maria Pires Bueno
Allyson Luiz De Cayres Lino
Evelin Vanessa Correia Dos Santos

Design instrucional e revisão textual

Fundação Getulio Vargas

Revisão técnica e pedagógica do *design instrucional*

Departamento de Educação Profissional e Educação de Jovens e Adultos

Ana Cristina Venancio Da Silva
Vanessa Raquel Dinho

Publicação

2017

SUMÁRIO

CARTA AO ALUNO	7
ESTRUTURA DO CONTEÚDO.....	8
CAPÍTULO 1: PROGRAMAÇÃO ORIENTADA A OBJETOS	9
TÓPICO 1: CONCEITUAÇÃO: OBJETOS, ATRIBUTOS, MÉTODOS E CLASSE斯.....	9
Introdução	9
Programação Orientada a Objetos (POO).....	9
Histórico.....	10
Programação Estruturada – Modelo Procedural	11
Comparação dos Modelos.....	13
Conceitos da Programação Orientada a Objetos.....	13
O que é um objeto?.....	14
Objetos.....	15
Atributos	15
Métodos	16
Desafio	17
Exemplo	18
Classes.....	18
Classe, Objeto, Atributo e Instância	20
Desafio	21
Estado do Objeto	21
Exemplo	22
Exercícios de Fixação.....	24
Encerramento do Tópico	25
TÓPICO 2: EVENTO E MENSAGEM.....	26
Evento.....	26
Parâmetros do Método.....	27
Exemplo	28
Interações entre Objetos.....	29
Interações entre Objetos	30
Mensagens	31
Exemplo	32
Exercícios de Fixação.....	33
Encerramento do Tópico e do capítulo	34
Síntese.....	35

CAPÍTULO 2: PRINCÍPIOS BÁSICOS DA ORIENTAÇÃO A OBJETOS..... 36

TÓPICO 1: ABSTRAÇÃO, ENCAPSULAMENTO E MODULARIZAÇÃO	36
Introdução	36
Pilares da Programação Orientada a Objetos	37
Abstração.....	37
Encapsulamento	38
Exemplo	40
Desafio	40
Modularização.....	41
Exemplo 1	41
Exemplo 2	42
Reutilização de Código.....	42
Exercícios de Fixação.....	43
Encerramento do Tópico	44
TÓPICO 2: POLIMORFISMO, HERANÇA, GENERALIZAÇÃO E ESPECIALIZAÇÃO	45
Polimorfismo	45
Reutilização de Código.....	46
Exemplo	46
Herança.....	47
Desafio	48
Generalização.....	49
Especialização	52
Exercícios de Fixação.....	53
Encerramento do Tópico e do capítulo	54
Síntese.....	55

CAPÍTULO 3: RELACIONAMENTO ENTRE CLASSES 56

TÓPICO 1: TIPOS DE RELACIONAMENTO – ASSOCIAÇÃO, AGREGAÇÃO E MULTIPLICIDADE	56
Introdução	56
Associação	56
Agregação	57
Multiplicidade.....	58
Relembrando Conceitos	59
Exercícios de Fixação.....	60
Encerramento do Tópico	61
TÓPICO 2: VANTAGENS DA PROGRAMAÇÃO ORIENTADA A OBJETOS	62
Vantagens da Programação Orientada a Objetos (POO)	62
Modelagem de um Sistema Orientado a Objetos.....	64
Relembrando Conceitos	65
Ética – Pense Nisso!	65
Exercícios de Fixação.....	67
Encerramento do Tópico e do capítulo	68
Síntese.....	69

LEITURA RECOMENDADA	70
REFERÊNCIAS BIBLIOGRÁFICAS	71
ENCERRAMENTO	72

CARTA AO ALUNO

Caro aluno,

Este conteúdo foi elaborado pensando em seu processo de aprendizagem. Nele você encontrará conceitos importantes sobre Programação Orientada a Objetos (POO), largamente utilizada em muitas linguagens de programação atuais, tais como Java, C #, PHP, Python, C++, entre outras.

Ao longo do estudo, você será convidado a fazer atividades e reflexões que contribuirão para seu desenvolvimento profissional na área de programação. Essas atividades são importantes para que você tenha uma visão teórica e prática da utilização dos conceitos de POO no desenvolvimento de sistemas. Fique atento a todo o conteúdo do curso para que você tenha um aprendizado significativo e condições de aplicar os conhecimentos em suas atividades.

Para alcançar os objetivos propostos, lembramos que sua dedicação e seu comprometimento são fundamentais. Leia o conteúdo com atenção, responda aos exercícios propostos, e aproveite as dicas e os recursos educacionais disponibilizados sobre os assuntos relacionados a sua área de atuação.

Ao término de seus estudos, você realizará uma avaliação para verificar sua compreensão a respeito dos assuntos abordados.

Desejamos a você um bom aprendizado!

ESTRUTURA DO CONTEÚDO

Este conteúdo está estruturado em três capítulos:

Capítulo 1: Programação Orientada a Objetos

- Tópico 1: Conceituação: Objetos, Atributos, Métodos e Classes
- Tópico 2: Eventos e Mensagens

Capítulo 2: Princípios Básicos da Orientação a Objetos

- Tópico 1: Abstração, Encapsulamento e Modularização
- Tópico 2: Polimorfismo, Herança, Generalização e Especialização

Capítulo 3: Relacionamento entre Classes

- Tópico 1: Tipos de Relacionamento: Associação, Agregação e Multiplicidade
- Tópico 2: Vantagens da Programação Orientada a Objetos

CAPÍTULO 1: PROGRAMAÇÃO ORIENTADA A OBJETOS

Tópico 1: Conceituação: Objetos, Atributos, Métodos e Classes

Neste tópico, vamos conhecer os conceitos relacionados à programação orientada a objetos (POO). Além disso, veremos a origem e a nomenclatura da POO, bem como as técnicas para o desenvolvimento de uma solução no âmbito da programação.

Conteúdos:

- Histórico
- Principais conceitos
- Objetos
- Atributos
- Métodos
- Classes.

Ao finalizar este tópico, você será capaz de:

- Identificar e classificar um objeto segundo a programação orientada a objetos.
- Identificar atributos e métodos que compõem a classe de um objeto.

Introdução

Antes de iniciarmos nosso estudo, imagine a seguinte situação:

Sua avó está fazendo bolinhos de chuva, mas acabou o açúcar!

Para ajudá-la, você vai ao supermercado comprar o ingrediente. No entanto, você precisa ser rápido, porque está com pressa para sair com os amigos.

Ao chegar no supermercado, você leva um susto. Todos os produtos estão misturados nas prateleiras – carnes ao lado de enlatados, biscoitos junto com detergentes... As prateleiras estão um verdadeiro caos!

Em um lugar assim, é possível encontrar o açúcar rapidamente? Claro que não!

Programação Orientada a Objetos (POO)

A programação orientada a objetos (POO) parte dessa mesma premissa do mundo real: é preciso organizar, categorizar e classificar objetos que tenham características em comum para ganharmos tempo. No caso da POO, ela organiza os objetos de acordo com suas classes, seus atributos e seus métodos. Desse modo, é possível facilitar a localização, a reutilização e a comunicação dos objetos.

Um supermercado organizado também funciona dessa maneira, não é mesmo?

Vamos entender melhor o conceito de programação orientada a objetos a partir de sua origem, sua nomenclatura e suas técnicas.

Nesse sentido, veremos como utilizamos a POO no desenvolvimento de uma solução, no âmbito da programação.

Para isso, precisamos compreender alguns conceitos. Vamos juntos?

A programação orientada a objetos (POO) é um modelo desenvolvido para aproximar o mundo real do mundo virtual, simulando o real no computador.

Nesse processo de programação, são criadas coleções de objetos com estrutura e comportamentos próprios. Tais objetos interagem entre si e executam as ações solicitadas.

A programação orientada a objetos é tida como a melhor solução para modelar e representar o mundo real em termos de escrita, em linguagem de programação.

Dessa forma, a POO é a interface entre a representação escrita de como nós pensamos e agimos, e de como a codificação e a CPU do computador vão entender e executar o que escrevemos.

Vamos descobrir como a POO surgiu?

Histórico

A primeira linguagem a utilizar os conceitos de orientação a objetos foi a [Simula 67](#)¹, mas o conceito de POO foi desenvolvido, no final da década de 1960, por [Alan Kay](#)².

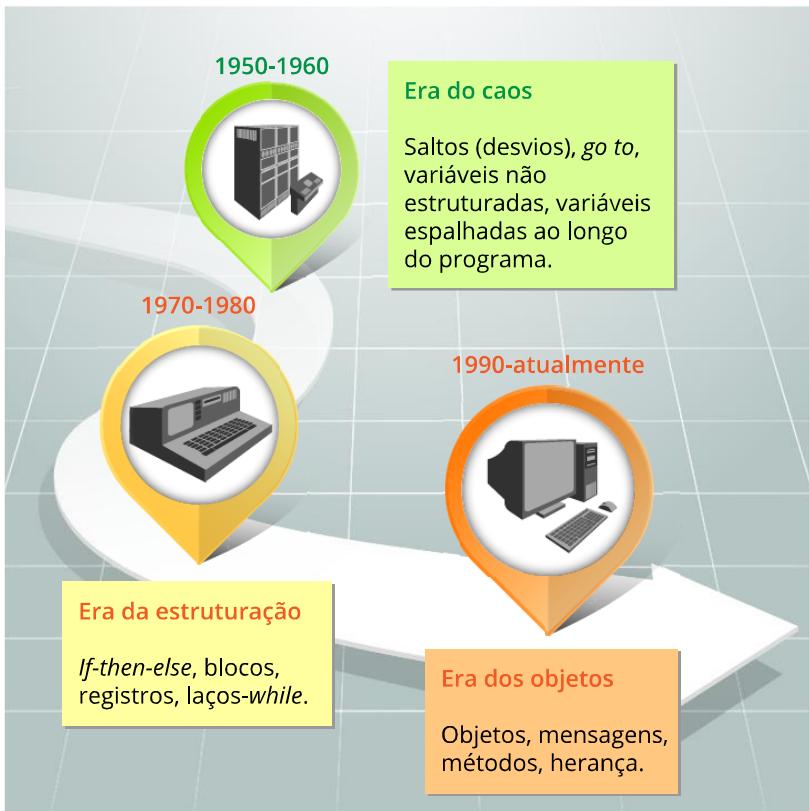
No entanto, somente na década de 1990, o modelo passou a ser adotado por grandes empresas e desenvolvedores de [software](#)³ do mundo.

1. Simula 67 – A Simula 67 foi uma linguagem criada por Ole Johan Dahl e Kristen Nygaard em 1967, na Noruega.

2. Alan Kay – Alan Kay foi autor da linguagem de programação chamada Smalltalk – Xerox. No entanto, a Smalltalk não foi a primeira linguagem a utilizar os conceitos de programação orientada a objetos.

3. Software – A palavra "software" está relacionada à toda codificação feita pelos desenvolvedores na composição de um aplicativo. De modo coloquial, chamamos o *software* de programa de computador.

Vamos observar as técnicas de programação utilizadas até a consolidação do POO:



Como podemos notar, o modelo de programação orientada a objetos é relativamente antigo.

Provavelmente, você já ouviu falar de algumas linguagens que utilizam o conceito de programação orientada a objetos, como Java, C# (C Sharp), C++, Object Pascal (Delphi), Ruby, Python, Lisp, entre outras.

Mas você já imaginou como os desenvolvedores programavam antes de surgir a linguagem orientada a objetos? Falaremos disso a seguir.

Programação Estruturada – Modelo Procedural

Antes do surgimento da programação orientada a objetos, o modelo adotado era o **procedural**. Esse modelo consistia em uma sequência de instruções, comandos, rotina, sub-rotina ou função associada a um nome próprio, como C, Pascal, BASIC, COBOL.

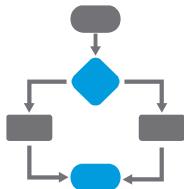
4. T-SQL – Transact-SQL (T-SQL) é propriedade da Microsoft e Sybase para a extensão SQL. A implementação da Microsoft foi emitida juntamente com o Microsoft SQL Server. A Sybase usa a linguagem no seu Adaptive Server Enterprise (ASE), que é o sucessor da Sybase SQL Server.

Entre alguns exemplos de aplicativos que podem gerar T-SQL (Transact-SQL) temos:

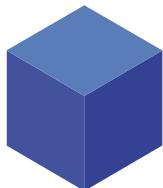
- Data warehouses, quando os dados são extraídos por meio de OLTP (Processamento de Transações On-Line) para análise de suporte à decisão.
- Sistemas que replicam dados do SQL Server para vários bancos de dados ou são executadas consultas distribuídas.
- Páginas da web que capturam informações de bancos de dados SQL Server.
- Aplicativos de produtividade para escritórios em geral.
- Aplicativos que usam uma interface gráfica do usuário com permissão de consulta às tabelas.

Vamos entender como a programação estruturada funcionava?

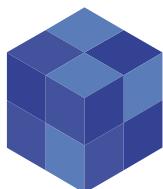
1. Os algoritmos e a linguagem Transact-SQL ([T-SQL⁴](#)) criavam procedimentos diversos.



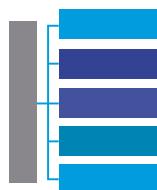
2. Tais procedimentos podiam ser decompostos em procedimentos menores e mais simples.



3. Esses procedimentos eram chamados de funções.



4. A execução do código era baseada na sequência de execução dos procedimentos, criando um emaranhado de operações, com procedimentos e funções ligados uns aos outros.



De modo geral, a execução do código por ordenação causava uma dependência enorme no processamento.

Além disso, gerava complexidade no código e dificuldade em futuras manutenções ou correções, pois não havia muito reutilização do código já escrito.

- Aplicativos que usam controle de linguagem para os dados que um usuário deseja consultar.
- Linha de aplicativos empresariais com acesso a bancos de dados SQL Server para o armazenamento de dados.
- APIs de banco de dados, como ADO, OLE DB e ODBC, que utilizam aplicativos de desenvolvimento como MicrosoftVisual C++, MicrosoftVisual Basic ou Microsoft Visual J++.

Fonte: Disponível em:
<https://msdn.microsoft.com/pt-br/library/bb510741.aspx>.
 Acesso em: 21 nov. 2016.

Sem um conhecimento profundo da codificação, o desenvolvedor preferia reescrever uma função, sem verificar se ela já existia ou se estava adequada ao que pretendia executar.

Na programação estruturada, o código acabava sendo repetido muitas vezes.

Comparação dos Modelos

Atualmente, a maioria das linguagens adota o conceito de programação orientada a objetos. No entanto, o desenvolvedor também está livre para escrever o código de modo procedural (tradicional) se preferir.

Vamos comparar a estrutura dos dois modelos de programação?



Conceitos da Programação Orientada a Objetos

Agora que já conhecemos os dois modelos de programação, vamos entender os principais conceitos da programação orientada a objetos.

Preparado(a)? Vamos lá!

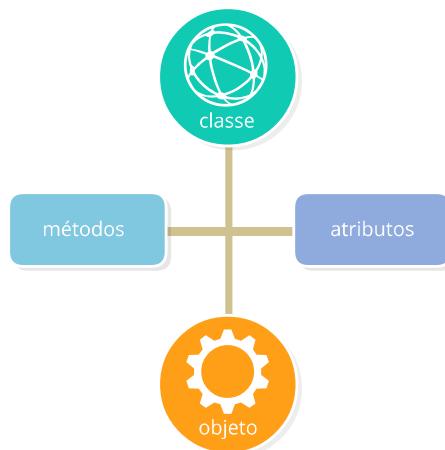
A programação orientada a objetos é um modelo de análise, de projeto e de programação de sistemas de *software* baseado na composição e na interação dos objetos. Vamos compreender melhor esse conceito.



Na programação orientada a objetos, é implementado um conjunto de **classes** que definem os **objetos** presentes no sistema de *software*.

Cada classe determina os comportamentos (**métodos**) e os estados possíveis (**atributos**) de seus objetos, bem como o seu relacionamento com outros objetos.

A seguir, veremos cada um desses conceitos.



O que é um objeto?

Como você certamente percebeu, a POO é um jeito de se pensar e de se organizar o código de programação dos *softwares*, cujo objetivo é possibilitar a reutilização de trechos de programação e, com isso, ganhar tempo e tornar a codificação mais “inteligente”.

Vídeo



Como o próprio nome sugere, os objetos são muito importantes neste tipo de programação e, justamente por isso, indicamos que você assista, no ambiente *on-line*, ao audiovisual **O que é um objeto?**, em que o professor Gustavo Guanabara explica e exemplifica os conceitos fundamentais da POO.

Objetos

Tal como você acabou de ver no vídeo indicado, os **objetos** são os elementos do mundo real que serão representados na programação do sistema de *software*. Para efeito prático, somente objetos relevantes para a solução do problema é que devem ser utilizados.

Vejamos alguns exemplos.

Objetos reais

Podem ser objetos reais: uma pessoa, um livro, um carro, um avião, um eletrodoméstico, um *notebook* etc.

Objetos abstratos

Podem ser objetos abstratos: as funções de pessoas nos sistemas – cliente, vendedor, representante, gerente, usuário etc.

Outros objetos

Podem ser considerados outros tipos de objetos:

- eventos – venda, compra, ligação
- interações com outros objetos – um item de nota fiscal é uma interação entre o objeto nota fiscal e um produto
- lugares – revenda, matriz, filial etc.

Atributos

Para entendermos o conceito de atributo, pensemos em um objeto bem comum em nosso dia a dia: o veículo, por exemplo.

Analisando o objeto Veículo, podemos verificar algumas características, tais como:

nome	modelo
fabricante	ano do modelo
cor	ano de fabricação
potência	combustível
código Renavam	chassi
nome do proprietário	número da placa
estado	tipo

As características que descrevem o objeto Veículo são também os valores do objeto. Esses valores são chamados de **atributos**.

Métodos

Já vimos o que são objetos e o que são atributos (características). Agora, está na hora de entendermos o que são métodos.

Os **métodos** equivalem às ações que o objeto é capaz de executar.

Na programação orientada a objetos, a única maneira de interagir com os objetos é por meio dos métodos (ações) que ele possui. Chamamos de **interface** o conjunto de métodos que um objeto possui.



A notação do método (ação) deve ser descrita como um verbo ou uma ação. Após a descrição, devem ser acrescentados parênteses.

Retomando o exemplo do carro, poderíamos dizer que os métodos são:

- acelerar()
- parar()
- transportar()
- ligar()
- desligar()
- buzinar()
- mudar de marcha()

Desafio

Antes de prosseguirmos, vamos ver se os conceitos de atributos e métodos estão claros. Para isso, preparamos um rápido desafio para você!

Considerando o objeto “livro”, classifique cada palavra listada abaixo como sendo um atributo ou um método:

	Atributo	Método
título	<input type="radio"/>	<input type="radio"/>
autor	<input type="radio"/>	<input type="radio"/>
cadastrar	<input type="radio"/>	<input type="radio"/>
consultar	<input type="radio"/>	<input type="radio"/>
gênero	<input type="radio"/>	<input type="radio"/>
alterar	<input type="radio"/>	<input type="radio"/>
ano	<input type="radio"/>	<input type="radio"/>
reservar	<input type="radio"/>	<input type="radio"/>

Comentário

Tal como dito anteriormente, os atributos são características (adjetivos) e os métodos, por sua vez, são sempre ações (verbos).

Exemplo

Agora, pense no objeto "celular". Você consegue listar, mentalmente ou em um rascunho, cinco atributos e cinco métodos desse objeto?

Veja alguns exemplos na tabela a seguir.

Atributos	Métodos
<ul style="list-style-type: none"> ▪ modelo ▪ marca ▪ cor ▪ peso ▪ dimensao_tela ▪ capacidade_memoria 	<ul style="list-style-type: none"> ▪ ligar() ▪ desligar() ▪ fazerchamada() ▪ receberchamada() ▪ enviarmensagem() ▪ acessaraplicativo()



Classes

Compare os dois carros a seguir:

		
Nome	Gol	Palio
Modelo	Confortline	EX
Fabricante	Volkswagen	Fiat
Ano de fabricação	2016	2014
Ano de modelo	2016	2014
Código Renavam	1234567890	1230987654
Proprietário	João da Silva	José da Silva
Cor	Vermelho	Azul
Potência	1.0	1.0
Combustível	Flex	Flex
Placa	EXE 1234	EXE 4321
Chassi	LKADSAJD3123DAS	HGKLFPG09765345
Estado	Ligado	Desligado
Tipo	Automóvel de passageiro	Automóvel de passageiro

Observe que os atributos dos objetos Gol e Palio são iguais, mas os valores são diferentes. Já os métodos podem ser iguais ou diferentes.

O que tentamos fazer no exemplo anterior foi uma classificação dos objetos Veículos. Para isso, colocamos esses objetos em uma ordem que nos possibilitasse entender cada um deles.

Ao ordenar os objetos, estamos entrando em um novo conceito – o conceito de classes.

Na programação orientada a objetos, uma classe define o comportamento dos objetos por meio de seus métodos e dos estados possíveis em que eles podem manter-se, devido a seus atributos.

Uma classe é composta de um conjunto de elementos que compartilham uma estrutura (atributos ou valores) e procedimentos comuns (métodos).

Desse modo, a classe representa um conjunto de objetos com características semelhantes.

Agora, vamos retomar os objetos Gol e Palio. Ambos são objetos do mundo real classificados como veículos. Dessa forma, a classe dos dois objetos pode ser considerada a mesma, se pensarmos na classe denominada Veículos.

Os carros possuem atributos ou valores que podem ser iguais ou diferentes.

Por exemplo:

- Estado: ligado/desligado
- Cor: prata
- Combustível: flex
- Métodos: acelerar(), frear(), transportar(), buzinar() etc.

Os objetos Gol e Palio são **instâncias⁵** da classe Veículos, cada um com seus atributos próprios.

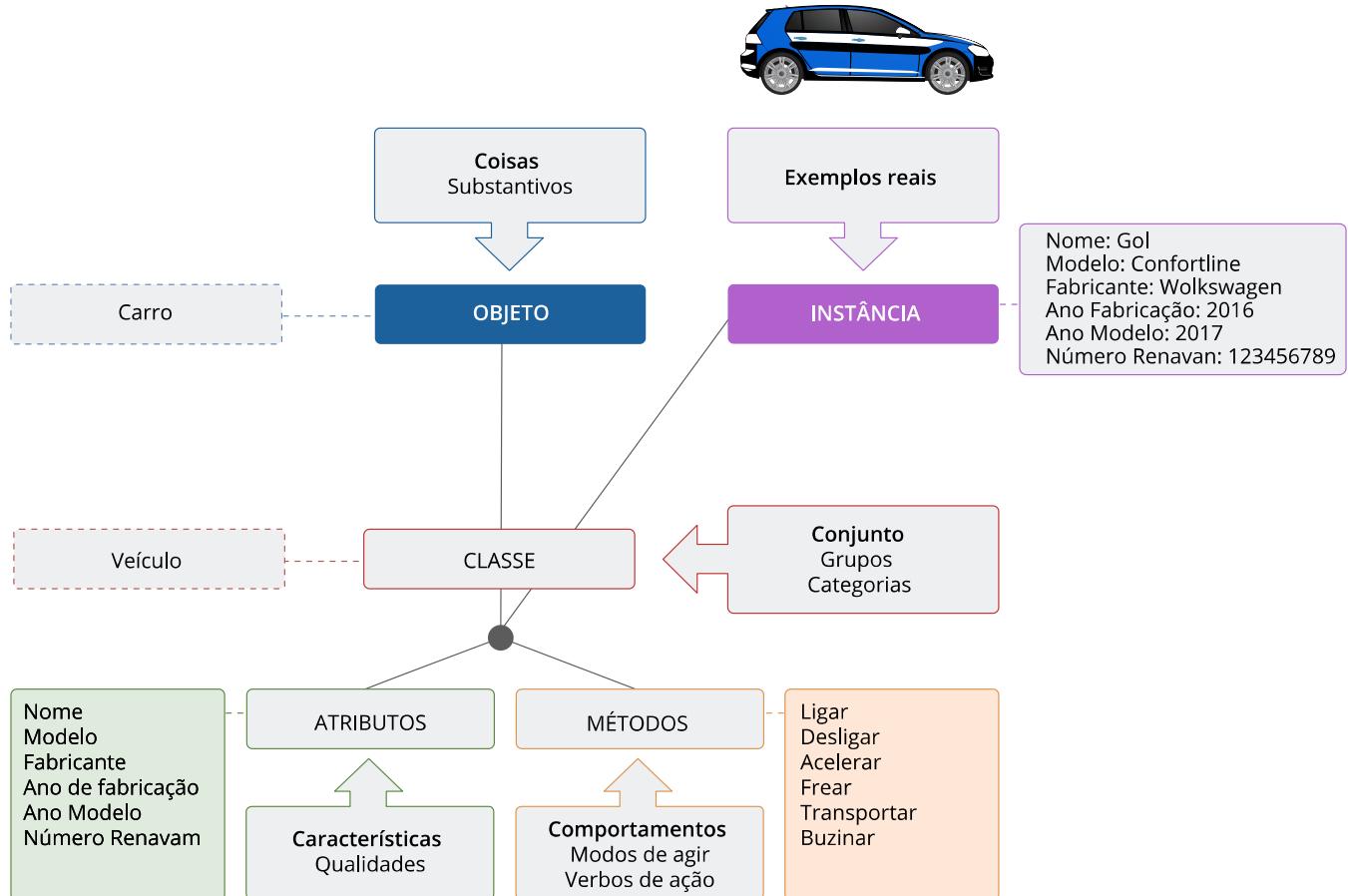
Gol e Palio são os objetos que representam a classe Veículos no mundo real. Dessa maneira, podemos afirmar que o **objeto é a instância de sua classe, com atributos próprios.**
Compreendeu?

5. Instâncias – Um objeto com atributos (valores) próprios é uma instância de uma classe.

Classe, Objeto, Atributo e Instância

Preparamos um infográfico que mostra a relação entre os conceitos de classe, objeto, atributo, método e instância.

Objetos, Classes, Atributos, Métodos e Instâncias



Fique atento! Atributos e métodos pertencem à classe. Já os valores dos atributos pertencem ao objeto.

Desafio

Além dos conceitos de atributos e métodos, queremos ver se você compreendeu os conceitos de classe e instância.

Vamos lá?

Complete as frases a seguir, selecionando o conceito que completa corretamente cada uma delas.

Um _____ (objeto/método) é a instância de sua classe. Atributos e métodos pertencem à classe.

Uma _____ (classe/instância) é um conjunto de atributos/valores e métodos comuns.

Estado do Objeto

Para entender o conceito de estado de objeto, vamos analisar um exemplo.

Primeiramente, imaginemos o elevador de um edifício. Em seguida, vamos pensar nos métodos (ações) e nos atributos (características) que podem ser atribuídos ao objeto Elevador.

Atributos	Métodos
<ul style="list-style-type: none">▪ Direção▪ Porta▪ Estado▪ Tipo de carga	<ul style="list-style-type: none">▪ transportar()▪ abrirPorta()▪ fecharPorta()▪ atenderChamado()

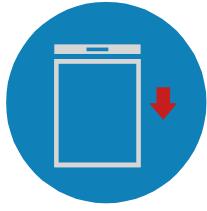
A partir do estabelecimento dos métodos e atributos, podemos chegar a algumas conclusões. Vejamos alguns exemplos.

Quando o elevador está subindo com pessoas:

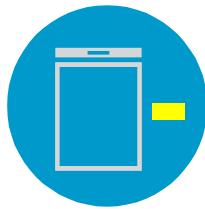


- Direção = Subindo
- Porta = Fechada
- Estado = Carregado
- Tipo de carga = Pessoas

Quando o elevador está descendo vazio:



- Direção = Descendo
- Porta = Fechada
- Estado = Vazio
- Tipo de carga = Pessoas



Quando o elevador está parado:

- Direção = Parado
- Porta = Aberta ou fechada
- Estado = Vazio ou carregado
- Tipo de carga = Pessoas

O objeto Elevador faz parte de uma classe também chamada de Elevador. Tal classe tem como método principal transportar.

Os atributos do objeto terão valores diferentes conforme o momento da operação em que esse objeto se encontra. Chamamos de **estado** o conjunto de valores dos atributos de um objeto em um dado momento.



Exemplo

Vamos analisar outro exemplo – o objeto chamado Connie.

A classe do objeto Connie será Cachorros. Nesse sentido, Connie tem nome, raça, cor, tamanho do pelo, porte, peso, idade, dono, moradia etc.

Nome: Connie

Raça: Sem raça definida

Cor: Predominantemente caramelo

Tamanho do pelo: Longo

Porte: Médio

Peso: 6 kg

Idade: 4 anos

Dono: Mariana

Moradia: Rua do Aconchego, 1234



Ao observar com atenção, podemos notar que:

- os **atributos** definem a **classe** Cachorros.
- os **valores** definem o **objeto** cachorro Connie.

Desse modo, os valores dos **atributos** serão diferentes para cada objeto Cachorro do mundo real. No entanto, todos os atributos farão parte da classe Cachorros. Vejamos:

Classe: Cachorros

Atributos: nome, raça, cor, tamanho do pelo, porte, peso, idade, dono, moradia.



objeto Connie



objeto Rex



objeto Tobi

Exercícios de Fixação

Agora, veja o quanto você sabe sobre o assunto tratado neste tópico. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Como técnico em desenvolvimento de sistemas, você foi incumbido de passar algumas informações sobre programação orientada a objetos aos novos estagiários da empresa.

Marque **V** para verdadeiro e **F** para falso em relação às informações sobre programação orientada a objetos que serão passadas a eles:

informações	V	F
A POO passou a ser adotada por grandes empresas e desenvolvedores de <i>software</i> do mundo somente a partir dos anos 2000.	<input type="radio"/>	<input type="radio"/>
A POO é tida como a melhor solução para modelar e representar o mundo real em termos de escrita em linguagem de programação.	<input type="radio"/>	<input type="radio"/>
A POO parte da premissa de que é preciso organizar, categorizar e classificar objetos que tenham características em comum para ganharmos tempo.	<input type="radio"/>	<input type="radio"/>
A POO organiza os objetos de acordo com suas classes, seus atributos e seus métodos, facilitando a localização, a reutilização e a comunicação dos objetos.	<input type="radio"/>	<input type="radio"/>

Questão 2

Você trabalha como programador JAVA em uma empresa de TI e a pedido de seu coordenador precisa preparar uma apresentação sobre POO para os novos programadores da empresa.

Selecione os termos apresentados entre parênteses para completar, corretamente, as afirmações a seguir, sobre os conceitos de POO.

- A. As características que descrevem o objeto são também os valores do objeto. Esses valores são chamados de _____ (atributos/dados).
- B. _____ (As funções/Os métodos) equivalem às ações que o objeto é capaz de executar.
- C. Chamamos de _____ (interface/classe) o conjunto de métodos que um objeto possui.
- D. _____ (Programa/POO) é um modelo desenvolvido para aproximar o mundo real do mundo virtual, simulando o real no computador.

Encerramento do Tópico

Neste tópico, entendemos melhor os conceitos relacionados à programação orientada a objetos a partir de sua origem, sua nomenclatura e suas técnicas.

Aprendemos também sobre os modelos existentes antes do surgimento da programação orientada a objetos.

Agora, já sabemos identificar e classificar um objeto segundo a programação orientada a objetos, bem como identificar atributos e métodos que compõem a classe de um objeto.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos? Siga para o próximo tópico!

Tópico 2: Evento e Mensagem

Neste tópico, vamos entender como os objetos, as classes, os atributos e os métodos interagem na lógica da programação orientada a objetos (POO).

Conteúdos:

- Evento
- Mensagens.

Ao finalizar este tópico, você será capaz de:

- Compreender como os objetos, as classes, os atributos e os métodos interagem entre si, de modo a compor um roteiro de desenvolvimento de codificação em qualquer linguagem.

Evento

Quando falamos em evento, em que, geralmente, pensamos?

Ao encontrarmos amigos em uma festa, por exemplo, temos um evento em que as pessoas estão interagindo, certo?

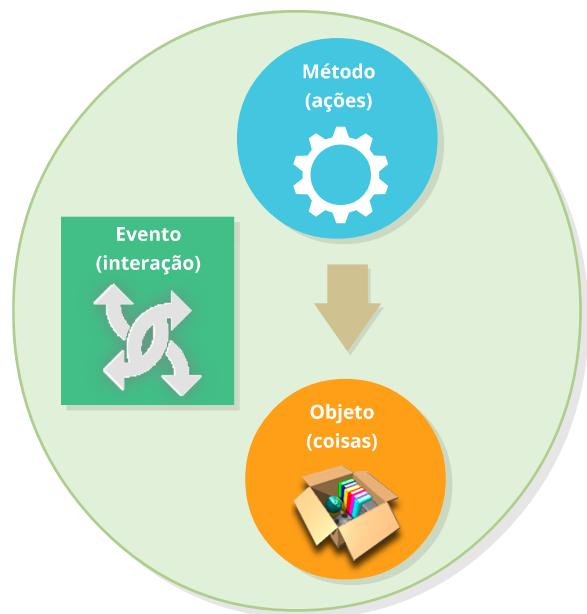
Evento é isso! Está relacionado a algo que acontece em determinado tempo.

Nesse sentido, um evento não tem nada a ver com a palavra "quando" – "quando algo vai acontecer."

No tópico 1, vimos que métodos são as ações (verbos) que o objeto pode realizar e que, na programação orientada a objetos, a única maneira de interagir com os objetos é por meio dos métodos que ele possui.

A princípio, um método é apenas uma definição de uma classe. A ação só ocorre quando o método é invocado por meio do objeto. Isso é chamado de evento.

Eventos são interações (parâmetros) internas e externas que auxiliam na execução do programa.



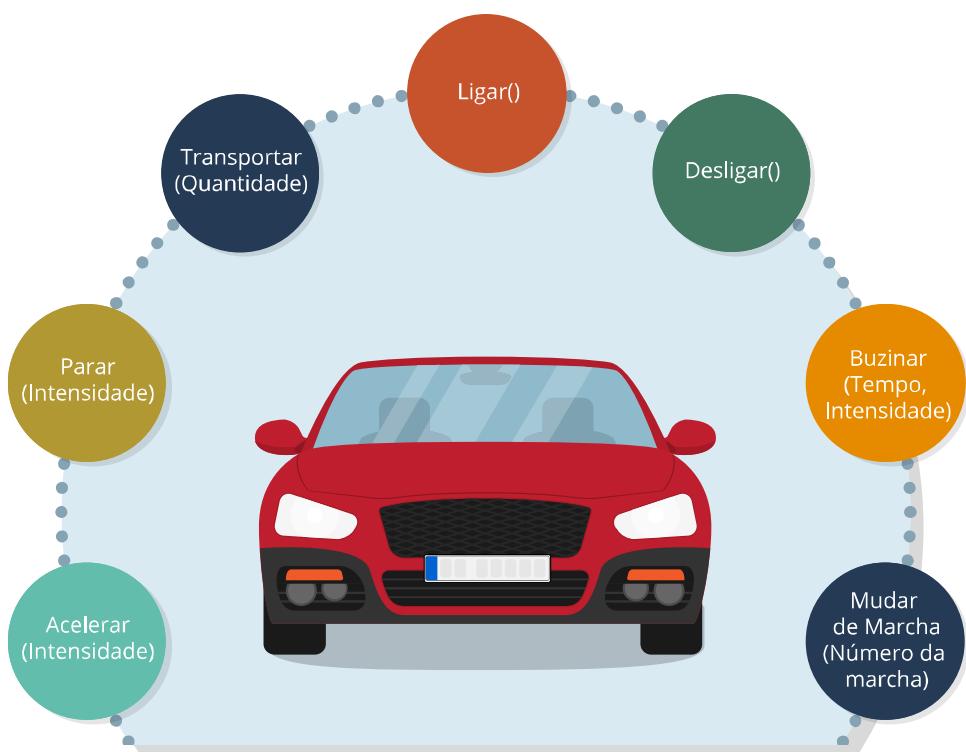
Parâmetros do Método

Um método pode alterar o atributo da classe com que está interagindo, por meio de argumentos em adição a seu objetivo (parâmetros). Isso significa que é possível passar parâmetros ou dados na chamada do método.

O método é parte da classe, e não parte do objeto. Nesse sentido, o método pode também ser herdado de outra classe. Em outro momento, veremos esse conceito de herança com mais detalhes.

O método é descrito por meio de um verbo, seguido de parênteses. Opcionalmente, podemos inserir os parâmetros do método entre os parênteses.

Vejamos alguns exemplos.



Exemplo

Vamos retomar o exemplo do objeto Connie. Sua classe continua sendo Cachorros. A classe Cachorros tem funções específicas – métodos específicos. Vejamos:



Desse modo, o que o cachorro Connie faz são os **métodos da classe Cachorros**. Os demais objetos Cachorros também têm os mesmos métodos.

Segundo a notação da UML⁶, a classe Cachorros é definida por seus atributos e seus métodos. Desse modo, teremos:

Classe: Cachorros

Atributos: nome, raça, cor, tamanho do pelo, porte, peso, idade, dono, moradia.

Métodos: latir(), comer(), dormir(), rosnar(), andar(), correr().

6. UML – A UML – *Unified Modeling Language* ou Linguagem Unificada de Modelagem – é a linguagem padrão aplicada para modelagem orientada a objetos.



objeto Connie



objeto Rex



objeto Tobi

Interações entre Objetos

Para compreender as interações entre objetos, vamos retomar o exemplo do elevador.

Quem inicia a operação do objeto Elevador é alguém que quer utilizar os métodos desse objeto, ou seja, as ações do elevador. Em outras palavras, tudo começa com alguém que necessita ser transportado a algum andar por meio do elevador.

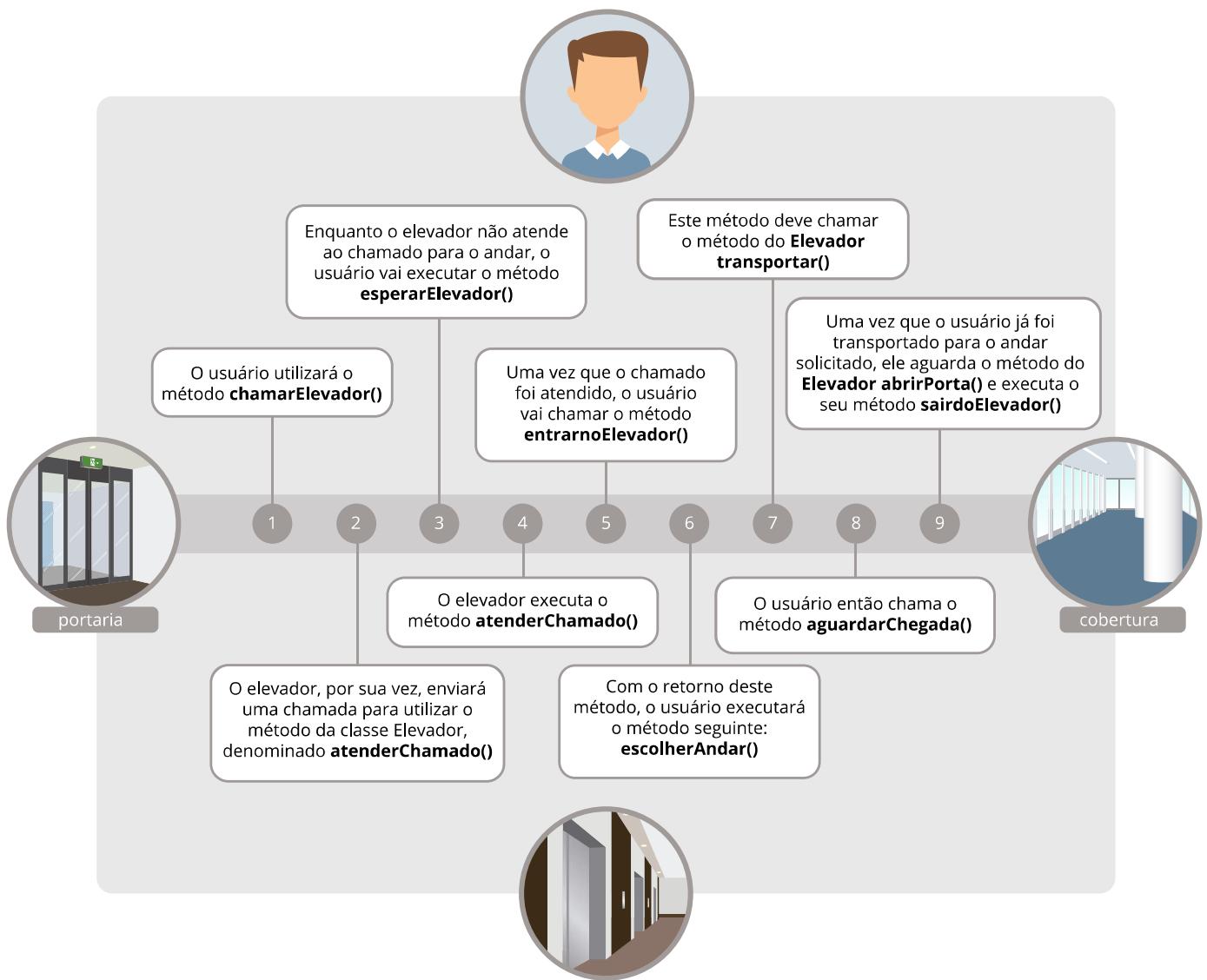
Nesse contexto, como o objeto João da Silva vai interagir com o objeto Elevador?

Como no mundo real, João vai chamar o elevador, esperar o elevador chegar, esperar a porta ser aberta, entrar no elevador e escolher para qual andar quer ir. Em seguida, João vai aguardar que o elevador chegue ao andar escolhido, esperar a porta ser aberta e sair.

Mas, como sequenciamos os métodos entre os objetos João da Silva e Elevador? Veremos isso a seguir.

Interações entre Objetos

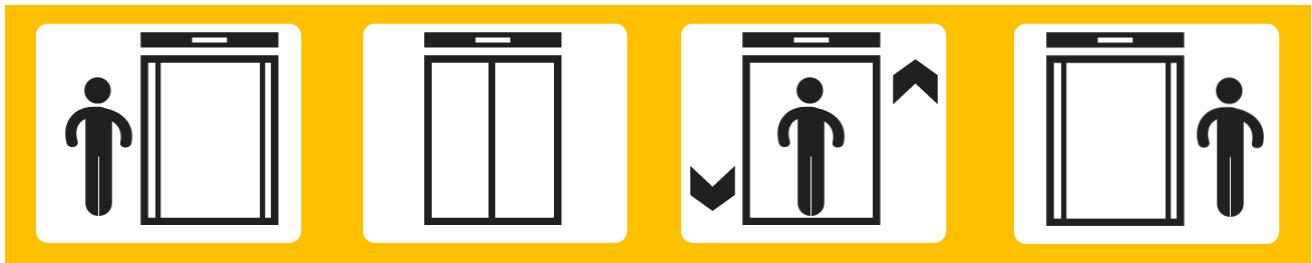
Para entender a sequência de ações executadas pelo usuário e pelo elevador, veja a ilustração a seguir.



O elevador é um tipo de objeto que sobe e desce carregando usuários o dia todo, não é mesmo? Isso significa que a sequência de métodos se repete indefinidamente na utilização do objeto Elevador pelo objeto Usuário.

Observe que a sequência de operação dos métodos deve seguir uma ordem específica. Mesmo estando com os objetos, os métodos e os atributos corretos, se a ordem for alterada, há o risco de a operação não funcionar.

Por exemplo, o usuário não pode utilizar o método sairDoElevador() caso o elevador não execute o método abrirPorta().



A complexidade nos relacionamentos entre os objetos sempre vai aumentar à medida que criamos mais classes para definir novos objetos. **Desse modo, precisamos definir, claramente, a que classe os métodos e os atributos devem pertencer.**

No exemplo simples do elevador, não estamos considerando todas as variações possíveis do mundo real.

Por exemplo, seria preciso considerar uma situação em que o elevador estivesse quebrado. Nesse caso, seria necessário estabelecer o que o usuário deve fazer. Ou ainda, como o elevador deve proceder se estiver lotado.

Quanto mais detalhamos e especificamos as situações, mais o código se aproxima do mundo real. Com isso, o código vai se tornando cada vez mais complexo.



Mensagens

Agora que compreendemos como ocorre a interação entre objetos, vamos ao último conceito deste capítulo: as **mensagens**.

Mensagens são trocas de informações entre os objetos. Tais informações são tratadas e devolvidas pelos métodos, quando necessário.

Vamos ver um exemplo?

Exemplo

Na classe Cachorros, como o objeto Connie vai executar os métodos que imaginamos?

Vamos conhecer como os métodos podem ser ativados.

- latir()
- comer()
- dormir()
- rosnar()
- andar()
- correr()

Quando estiver na presença de outro objeto.

Quando estiver com fome.

Quando estiver cansado.

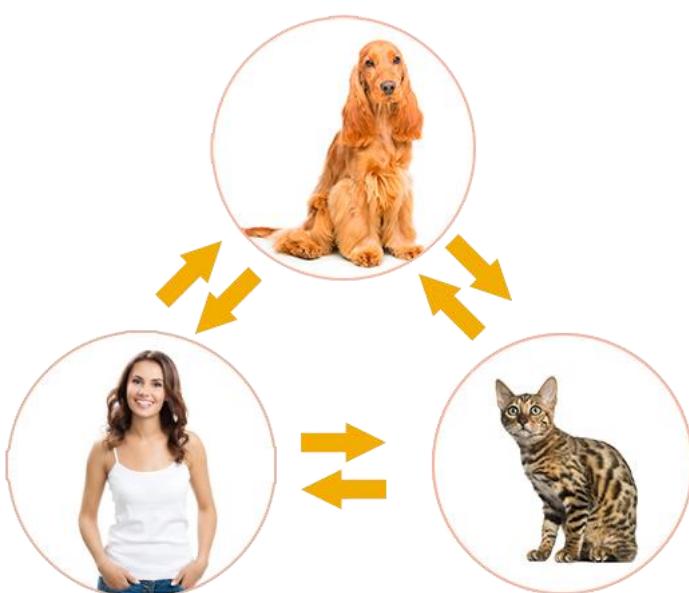
Quando estiver na presença de outro objeto Cachorro ou de um objeto Gato.

Quando estiver na companhia do dono ou sozinho.

Quando estiver com vontade de brincar.

mensagens

Os estímulos listados são as mensagens. Com isso, podemos afirmar que as mensagens são criadas pelas interações dos objetos com outros objetos. Vejamos:



Exercícios de Fixação

Agora, veja o quanto você sabe sobre o assunto tratado neste tópico. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Como especialista em POO, você foi chamado para fazer uma apresentação sobre essa tecnologia e precisa resumir, em poucas frases, as interações entre objetos utilizando, como exemplo, os objetos elevador e usuário.

Assim, você iniciou a sua apresentação destacando que o elevador é um tipo de objeto que sobe e desce carregando usuários o dia todo. Isso significa que a sequência de métodos se repete, indefinidamente, na utilização do objeto elevador pelo objeto usuário.

Dessa forma, pode-se observar que a sequência de operação dos métodos segue uma ordem específica, mesmo estando com os objetos, os métodos e os atributos corretos. Se a ordem for alterada, há o risco de a operação não funcionar.

Indique os termos que completam as definições apresentadas:

TERMOS

- (1) CLASSE
- (2) OBJETOS
- (3) OPERAÇÃO
- (4) MÉTODOS

DEFINIÇÕES

- A. Precisamos definir, claramente, a qual _____ os métodos e os atributos devem pertencer.
- B. Quanto mais detalhamos e especificamos a _____, mais o código se aproxima do mundo real.
- C. A sequência de _____ se repete indefinidamente, na utilização do objeto Elevador pelo objeto Usuário.
- D. A complexidade nos relacionamentos entre os _____ sempre vai aumentar à medida que criamos mais classes.

Questão 2

Você recebeu um novo estagiário para compor a sua equipe e explicou a ele que métodos são ações que o objeto pode realizar e que, na programação orientada a objetos, a única maneira de interagir com os objetos é por meio dos métodos que ele possui. Complementando as informações, sugeriu a ele que estudasse a respeito do método POO.

Agora, indique os termos entre parênteses que completam as afirmações a seguir, a respeito de método em POO que você deverá trabalhar com o estagiário:

- A. O método pode ser herdado de _____ (outra classe/outro atributo).
- B. É _____ (impossível/possível) passar parâmetros ou dados na chamada do método.
- C. _____ (Obrigatoriamente/Opcionalmente), podemos inserir os parâmetros do método nos parênteses.
- D. Um método pode alterar o atributo da classe com o qual está interagindo por meio de _____ (argumentos/objetos) em adição a seu objetivo.

Encerramento do Tópico e do capítulo

Neste tópico, vimos que os objetos, os atributos e os métodos interagem em eventos, a partir das mensagens.

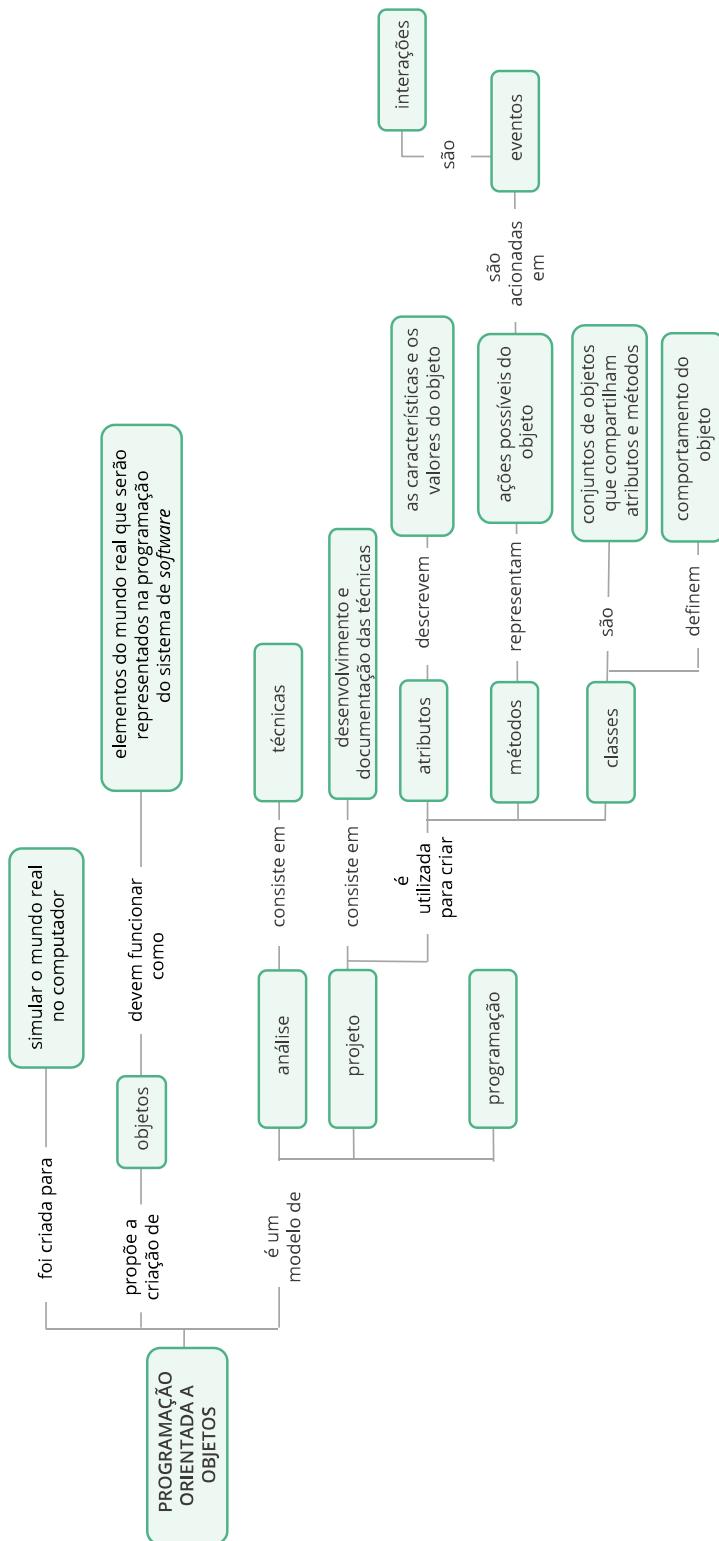
Agora, você já sabe como os objetos, as classes, os atributos e os métodos interagem entre si, de modo a compor um roteiro de desenvolvimento de codificação em qualquer linguagem.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Você está prestes a concluir o capítulo. No ambiente *on-line*, para facilitar seus estudos, disponibilizamos um PDF com um resumo dos conteúdos abordados neste capítulo, que se encontra na próxima página.

Síntese

Observe, a seguir, o mapa conceitual que sintetiza o conteúdo que acabamos de trabalhar:



CAPÍTULO 2: PRINCÍPIOS BÁSICOS DA ORIENTAÇÃO A OBJETOS

Tópico 1: Abstração, Encapsulamento e Modularização

Neste tópico, veremos como a programação orientada a objetos trata as tarefas repetitivas, bem como de que modo um código já escrito pode ser reutilizado e melhorado. Nesse sentido, vamos entender como a modularização pode deixar o código mais seguro e confiável.

Conteúdos:

- Abstração
- Encapsulamento
- Modularização.

Ao finalizar este tópico, você será capaz de:

- Compreender o conceito de encapsulamento e entender o quanto ele é necessário.
- Compreender os conceitos relacionados à modularização ou componentização.

Introdução

Antes de iniciarmos nosso estudo, imagine a seguinte situação:

Imagine que você tenha sido requisitado para cobrir as férias de um colega programador.

Logo no primeiro dia, pedem que você crie um novo módulo para um *software* que está sendo desenvolvido.

Há dois caminhos para encarar essa situação:

- recomeçar a ler toda a documentação e programar tudo do zero;
- reutilizar os módulos já criados.

O caminho mais econômico é reutilizar o que já foi criado.

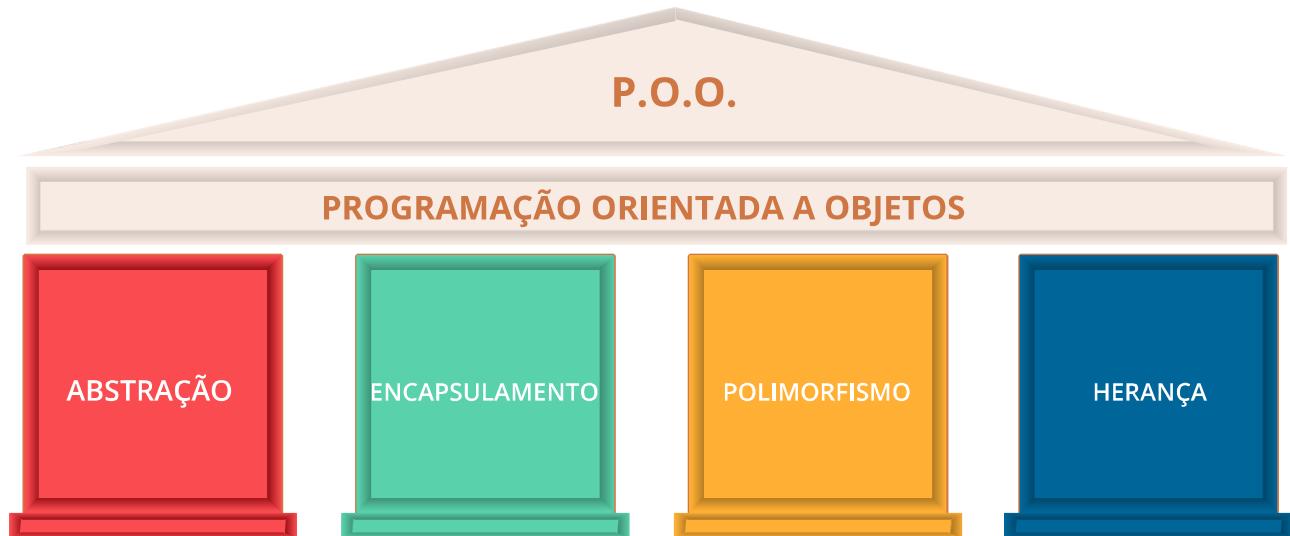
Para isso, é preciso que seu colega tenha escrito tudo de forma modularizada.

A escrita modularizada é um dos pilares da orientação a objetos.

Que tal conhecer todos os pilares da POO? Vamos lá!

Pilares da Programação Orientada a Objetos

Vamos conhecer os quatro pilares da programação orientada a objetos.



Abstração

A abstração consiste na habilidade de se concentrar nos aspectos essenciais de um contexto qualquer, possibilitando a modelagem de conceitos, elementos, problemas e características do mundo real em um domínio do sistema de *software*.

No processo de abstração, as características menos importantes ou acidentais são desconsideradas. Nessa lógica, apenas os detalhes importantes para a resolução do problema são levados em conta. Em outras palavras, os detalhes que não tem importância no contexto são desprezados.

A abstração nem sempre produz os mesmos resultados. Desse modo, os resultados da abstração dependem do contexto em que ela é utilizada.

A abstração de um objeto em um contexto pode ser diferente de outro.

Vamos entender melhor a partir de um exemplo?

Como vimos, a abstração é a habilidade de contextualizar problemas e características do mundo real, transpondo-os para o domínio dos sistemas de *software* e ignorando aspectos que não façam parte do contexto desenvolvido.

Vejamos um exemplo de abstração do objeto Carro.

Para um mecânico, o carro é caracterizado por marca, modelo, potência, cilindradas, motor, peças, defeitos, problemas etc.

Para o proprietário, o carro é caracterizado por cor, marca, modelo, potência, preço, ano, consumo, *status* etc.

Para o Departamento de Trânsito (Detran), o carro é caracterizado por placa, marca, modelo, proprietário, cor, chassi, IPVA, seguro obrigatório, licenciamento etc.

Ficou mais claro? Vamo-nos aprofundar um pouco mais nesse assunto!

A abstração requer que analisemos um objeto sob diferentes ângulos.

Imagine uma Pet Shop. Os atributos e os métodos da loja variam de acordo com o ponto de vista do:



Nesse sentido, a abstração permite constatar que os objetos e os métodos variam de acordo com o **referencial**.

Para que um sistema seja completo, é importante que façamos o exercício da abstração. Desse modo, poderemos prever o maior número de atributos e métodos necessários ao bom funcionamento desse sistema.

Agora que já vimos o pilar da abstração, vamos passar para o encapsulamento!

Encapsulamento

A cápsula existe para proteger algo, certo? Um medicamento, por exemplo.

Na orientação a objetos, a cápsula tem a mesma função.

O encapsulamento é um mecanismo que permite proteger métodos e atributos de uma classe.

Por meio desse mecanismo, podemos impedir o acesso direto por outros objetos e garantir que todo o processo de comunicação com o objeto aconteça por um conjunto predeterminado de operações.

A proteção oferecida pelo encapsulamento é baseada na utilização de modificadores de acesso (*public*, *private*, *protected*, *default*) – mais limitante sobre os atributos aplicados na classe –, disponibilizando métodos internos que modificam os valores dos atributos da classe declarada.

A partir dos modificadores de acesso, o encapsulamento serve para **controlar o acesso aos atributos e métodos de uma classe**.

Dentro de uma classe, os atributos podem ser declarados públicos e privados.



Além de facilitar a manutenção de classes e a integridade dos atributos de um objeto por certo momento, o encapsulamento ajuda a prevenir o problema da interferência indevida de um objeto externo nos dados de outro objeto.

Em outras palavras, o encapsulamento é o mecanismo que permite a proteção de métodos e atributos sensíveis dentro de uma classe, impedindo o acesso direto a eles por outros objetos. Vamos ver um exemplo?

O usuário de um *site* de banco somente pode verificar seu saldo, mas não pode modificá-lo.

A modificação de saldo só pode ser feita pelo sistema do banco, mediante lançamento contábil de débito ou crédito.

Nesse caso, **usuário** e **senha** são dados sensíveis que devem ser **protegidos** de acesso indevido.

Se eles forem declarados como dados públicos erroneamente, qualquer objeto criado para uso no desenvolvimento de um *software* poderá "enxergar" tais dados.

Esse tipo de erro pode ser catastrófico em termos de segurança.

Exemplo

Vejamos mais um exemplo. Imagine que você está trabalhando para um hospital.

Observe os objetos da cena a seguir e pense em quais dados sensíveis desse contexto deveriam ser preservados. Em seguida, vejamos alguns exemplos.

	Médico Histórico de trabalho e dados pessoais – nome, endereço, telefone, número de documentos.	Funcionária Salários e dados pessoais – nome, endereço, telefone, número de documentos.	Paciente Prontuário médico, resultados de exames, histórico de pagamentos e dados pessoais – nome, endereço, telefone, número de documentos.
--	---	---	--

Desafio

Antes de prosseguirmos para outro pilar da orientação a objetos, vamos a mais um desafio?

Quais seriam os dados sensíveis de um relacionamento entre cliente e banco? Veja alguns exemplos de respostas:

Dados pessoais:

- Nome
- Endereço de entrega
- Telefone.

Movimentação bancária:

- Saldo da conta
- Créditos
- Débitos
- Senhas.

Modularização

A modularização (ou **componentização**⁷) é o mecanismo que permite que um sistema de *software* seja dividido em partes que interagem entre si. Tais partes são chamadas de módulos.

Desse modo, o código desenvolvido é dividido em módulos independentes, que podem ser utilizados por qualquer objeto e a qualquer tempo.

7. Componetização – Alguns autores chamam a modularização de componentização. O termo representa a divisão do sistema em componentes ou módulos.

A modularização permite que uma parte do código possa ser alterada ou atualizada sem que todo o código já desenvolvido seja alterado. Dessa maneira, fica muito mais fácil atualizar um *software* ou corrigir um problema específico em um sistema.

Em outras palavras, podemos agrupar classes com funções e métodos semelhantes em um mesmo módulo. Tal módulo é acoplado a outros módulos (métodos) para a execução das tarefas solicitadas (mensagens).

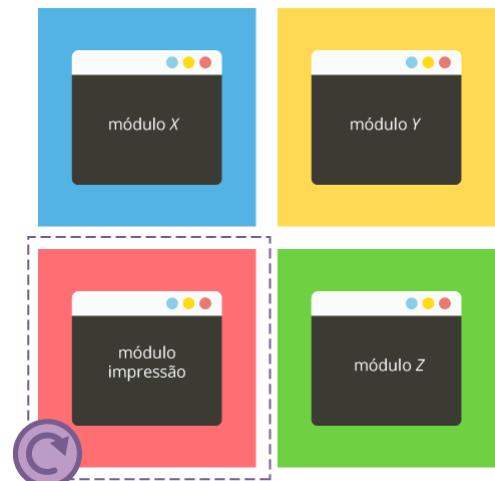
Vamos ver alguns exemplos?

Exemplo 1

Como vimos, a modularização é o mecanismo de programação que permite que o desenvolvedor separe em módulos as classes que têm métodos e atributos com funções semelhantes.

Para exemplificar, vamos usar o método de uma classe chamada Impressão.

A classe Impressão está contida no módulo de impressão do *software*. Imagine que um erro de impressão foi descoberto e corrigido. Com isso, o código foi remontado – recompilado.

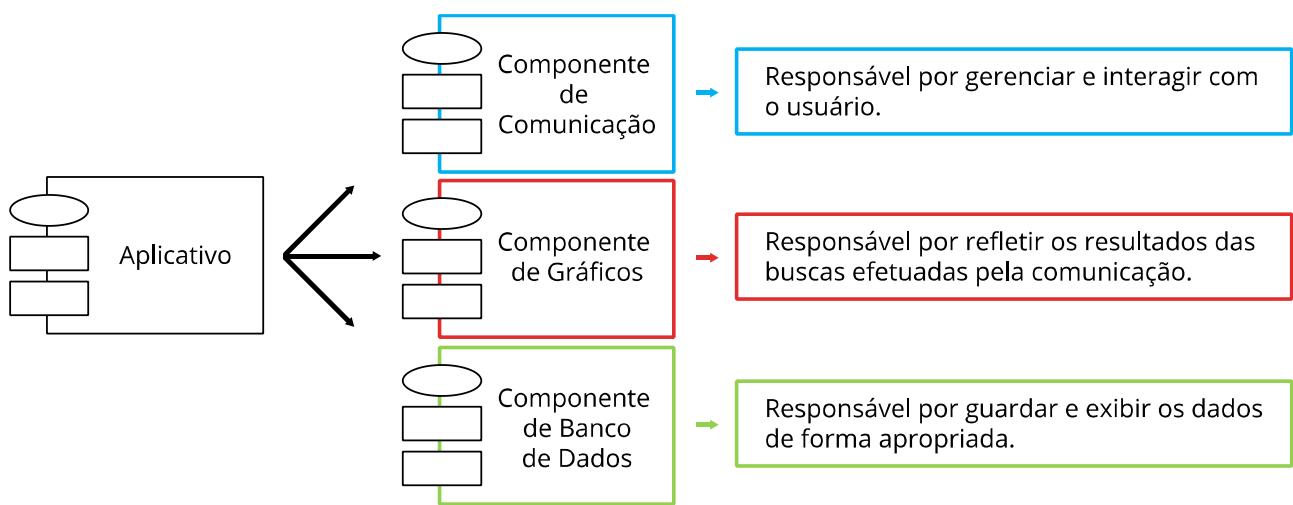


Nesse caso, basta atualizar o módulo de impressão para que a correção tenha efeito.

Exemplo 2

O Sistema Operacional Windows utiliza muitos módulos chamados de DLLs – *Dynamic-Link Libraries* ou bibliotecas de vínculo dinâmico. Desse modo, quando fazemos uma atualização no sistema operacional, as DLLs são trocadas.

O desenvolvedor também pode encapsular suas classes em DLLs. Quando é feita a atualização de uma DLL específica em parte do *software*, a atualização ocorre a partir da substituição dessa DLL. Para ilustrar esse exemplo, veja uma representação gráfica da modularização:



Repare que separamos o aplicativo em três módulos principais – Comunicação, Gráficos e Banco de Dados. Cada módulo cuida, especificamente, da parte que lhe compete.

Reutilização de Código

Antes de encerrarmos o tópico, vamos aprender a reutilizar códigos por meio de um exemplo.

Imagine que você esteja desenvolvendo um sistema para uma empresa de turismo. O sistema deve englobar a parte de hotelaria e de reserva de passagens. Será que é possível aproveitar a codificação de um sistema para o outro?

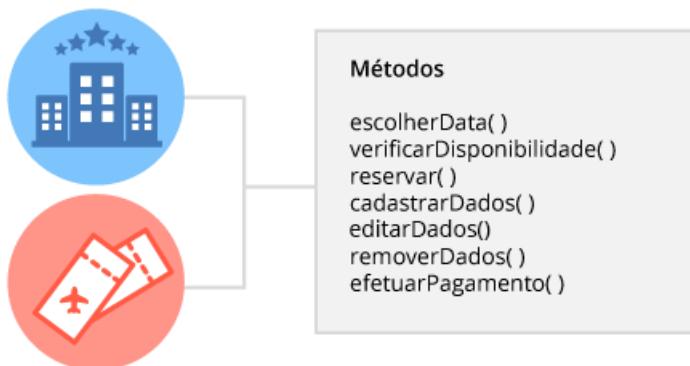
Partindo do conceito de modularização, já sabemos que o *software* pode ser dividido em partes ou módulos independentes. Com isso, é possível realizarmos a atualização e a correção de problemas específicos.

No entanto, repare que há uma vantagem que vai além da resolução de problemas. O maior ganho com a modularização está na melhoria do fluxo do programa e na viabilidade de reuso dos códigos.

Se é viável reutilizar códigos, como podemos fazer isso na prática? Vamos descobrir!

Primeiramente, temos de encontrar as similaridades entre os sistemas e definir os métodos em comum. A partir daí, poderemos criar os módulos.

Vamos analisar as similaridades:



Observe que alguns métodos podem ser utilizados em ambos os sistemas. Desse modo, os métodos comuns poderão ser reutilizados com a modularização.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre o assunto tratado neste tópico. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Você trabalha com o desenvolvimento de sistemas que utilizam programação orientada a objetos: um padrão de desenvolvimento – seguido por muitas linguagens, com Java e C# – que tem evoluído muito e está assentado em quatro grandes pilares, destacando questões voltadas para segurança e reaproveitamento de código.

Esses pilares são:

- UML, POO, Classes e Objetos.
- Classes, Objetos, Relacionamentos e Multiplicidade.
- Abstração, Encapsulamento, Polimorfismo e Herança.
- Polimorfismo, Herança, Generalização e Especialização.

Questão 2

Você está participando de um processo seletivo para a vaga de analista de programador de POO de uma grande empresa de tecnologia da informação e, na parte teórica do processo seletivo, teve de distinguir algumas informações em relação aos diagramas da UML.

Marque **V** para verdadeiro e **F** para falso em relação aos diagramas da UML.

informações	V	F
Motor, cor, chassi e marca sempre serão atributos de um objeto carro.	<input type="radio"/>	<input type="radio"/>
Para agilizarmos o desenvolvimento dos sistemas, é uma boa prática definirmos os dados de <i>login</i> de um usuário como atributos públicos.	<input type="radio"/>	<input type="radio"/>
A divisão dos sistemas POO em módulos ou componentes permite a independência das partes, facilitando a manutenção ou correção de problemas, já que podemos atuar em uma parte sem precisar mexer no todo.	<input type="radio"/>	<input type="radio"/>
O maior ganho com a modularização está na melhoria do fluxo do programa e na viabilidade do reuso de códigos.	<input type="radio"/>	<input type="radio"/>

Encerramento do Tópico

Neste tópico, vimos como a programação orientada a objetos trata as tarefas repetitivas. Também aprendemos como reutilizar e melhorar um código já escrito.

Agora, você já comprehende a importância do conceito de encapsulamento e dos conceitos relacionados à modularização (ou componentização).

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos? Siga para o próximo tópico!

Tópico 2: Polimorfismo, Herança, Generalização e Especialização

Neste tópico, vamos ver como o polimorfismo reaproveita o código já criado e o transforma em outra proposta. Além disso, vamos entender como a herança representa um ganho real na programação.

Conteúdos:

- Polimorfismo
- Herança
- Generalização
- Especialização.

Ao finalizar este tópico, você será capaz de:

- Compreender o conceito de polimorfismo e suas aplicações.
- Compreender o conceito de herança e suas aplicações.
- Aplicar os conceitos relacionados à programação orientada a objetos para escrever códigos.

Polimorfismo

Agora, vamos conhecer o terceiro pilar da programação orientada a objetos: o **polimorfismo**.

Na POO, o polimorfismo denota uma situação em que um objeto pode comportar-se de maneiras diferentes ao receber uma mensagem. O comportamento do objeto vai depender do modo como ele foi concebido.

O polimorfismo é complementado pelos conceitos de **herança** e **sobrecarga** de métodos.

O conceito de herança será abordado em seguida. Já a sobrecarga consiste em escrever métodos de mesmo nome com assinaturas diferentes. Em outras palavras, podemos criar vários métodos de mesmo nome com diferentes passagens de parâmetros.

Vamos entender melhor a sobrecarga por meio de um exemplo?

Imagine um aplicativo que faça a impressão de relatórios por meio de uma classe chamada Impressão.

O aplicativo é uma interface de acesso às funcionalidades da impressora usada pelo usuário e funciona por meio de um *driver*⁸ fornecido pelo fabricante. Cada fabricante disponibiliza o *driver* de sua impressora.



8. Driver – No contexto da informática, a palavra "driver" se refere a um conjunto de códigos escritos (instruções) que fazem a comunicação entre o sistema operacional de sua máquina e o *hardware*.

Sabemos que uma impressora a *laser* tem um mecanismo de impressão totalmente diferente de uma impressora a jato de tinta. No entanto, para o aplicativo, isso não importa, pois o envio da mensagem à impressora para que ela possa imprimir é feito por meio de códigos ou instruções.

Como vimos, a maneira como a impressora imprime o relatório varia de acordo com o tipo do equipamento. Nesse caso, a impressão ocorre de formas diferentes para a mesma mensagem de impressão.

Reutilização de Código

Outro exemplo de sobrecarga de método ocorre quando enviamos o comando **imprimir** a partir de um documento do Word.

Nessa situação, abre-se uma caixa de diálogo onde o usuário deve selecionar para qual impressora será dada a função de imprimir.

Ao abrir um documento do Excel e solicitar sua impressão, a mesma caixa de diálogo aparecerá. Você sabe como o módulo de impressão sabe a diferença entre um documento do Word, uma planilha do Excel, uma imagem e uma fotografia?

Para que um documento visualizado na tela do computador se torne um documento impresso, muitos métodos, classes, interfaces e códigos foram escritos. A partir daí, a operação **imprimir** se torna extremamente simples.

Vamos adiante!

Exemplo

Vejamos um exemplo de polimorfismo para o método **comunicar()**?

Observe abaixo como diferentes objetos respondem ao método (ação) **comunicar()**:



Note que os diferentes objetos – criança, cão, gato e pássaro – comunicam-se de diferentes formas. Desse modo, as respostas do método **comunicar()** são diferentes para cada um dos objetos.

Isso significa que classes comuns podem conter mensagens diferentes se os objetos utilizarem essas classes para um método comum.

Herança

Vimos que o polimorfismo é complementado também pelo conceito de **herança** – o quarto e último pilar da POO.

A palavra "herança" significa aquilo que se herda, aquilo que é transmitido por hereditariedade.

Podemos entender a herança por meio da genética. Por exemplo, um filho herda as características genéticas dos pais e possui suas próprias características. Por sua vez, repassa suas características a seus filhos. Desse modo, os filhos do filho podem ter características dos avós e as próprias características também.

A notação de uma herança é representada por um triângulo com o vértice apontado para a classe base.

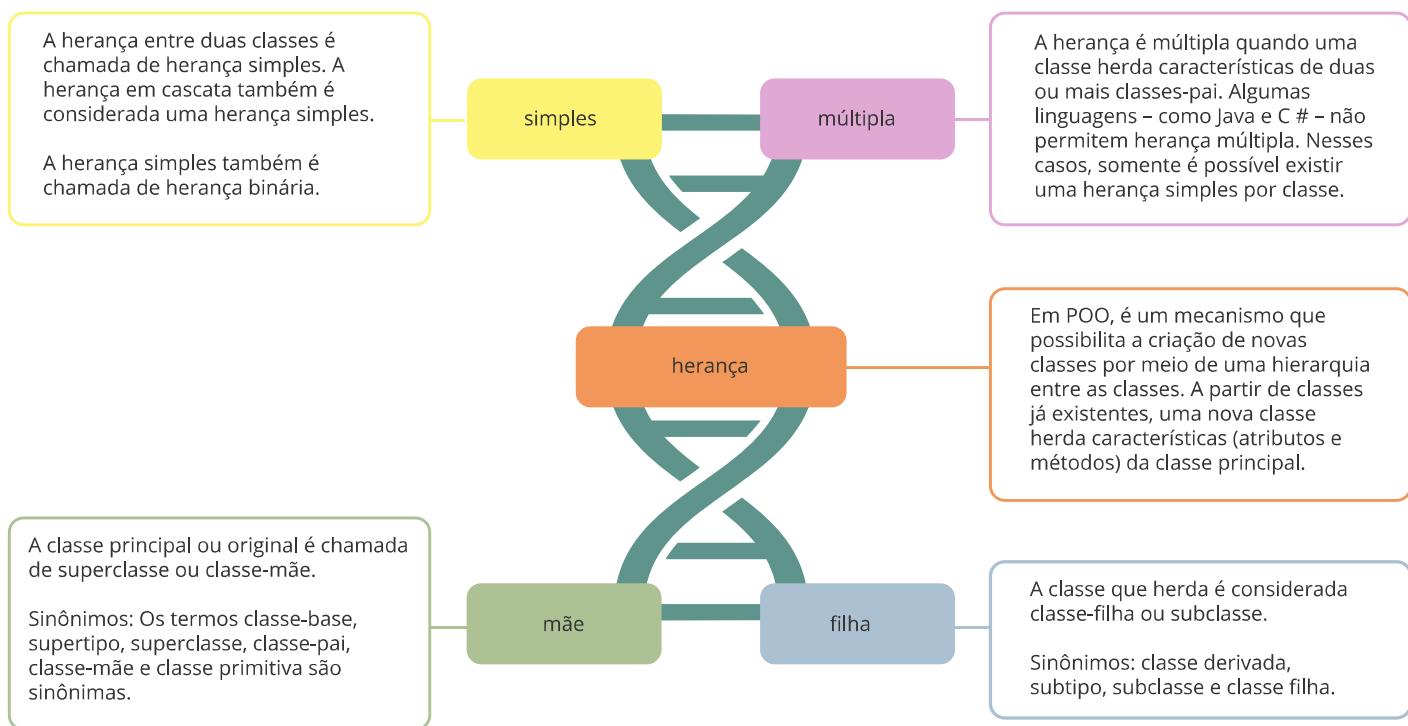
Outra notação muito aceita é a seguinte: coloca-se o nome da classe filha mais o símbolo de dois-pontos (:) e, em seguida, coloca-se o nome da classe herdada, como podemos observar no esquema a seguir.



Mas como o conceito de herança se aplica na POO? É o que veremos a seguir.

Para entender a aplicação do conceito de herança na programação orientada a objetos, veja o infográfico a seguir.

HERANÇA EM PROGRAMAÇÃO ORIENTADA A OBJETOS



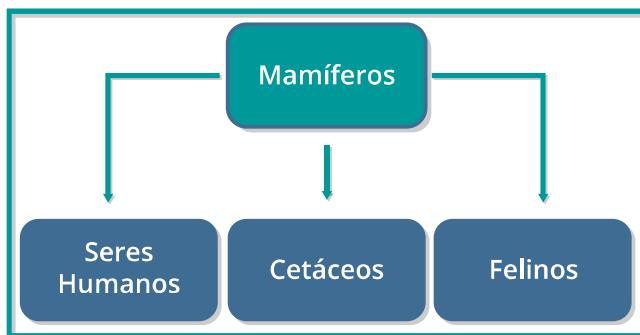
Desafio

Como vimos, o conceito de herança estabelece que uma classe pode herdar características de outra. A partir disso, propomos a você um desafio!

Observe o esquema a seguir. Em seguida, tente imaginar o que pode ser herdado da classe **mãe** (Mamíferos) para as classes filha (Seres Humanos, [Cetáceos⁹](#) e Felinos).

9. Cetáceos– Animais mamíferos marinhos. Por exemplo, as baleias.

Pense em dois atributos e dois métodos comuns pelo menos. Vejamos exemplos de resposta.



De acordo com o exemplo, algumas características que podem ser herdadas da classe Mamíferos pelas classes Seres Humanos, Cetáceos e Felinos são:

Atributos comuns:

- Idade
- Sexo

Métodos comuns:

- alimentar()
- procriar()
- dormir()

Generalização

Segundo o conceito de generalização, a classe filha pode estender ou repassar seus métodos e atributos para outra classe – classe mãe. Em outras palavras, novas classes são criadas por meio da similaridade entre as classes. Vejamos um exemplo de generalização.

Inicialmente, vamos criar uma classe chamada **Animais**. Os métodos e atributos da classe Animais serão:

Animais	
Métodos	Atributos
<ul style="list-style-type: none"> ▪ andar() ▪ alimentar() ▪ dormir() ▪ procurarAlimento() ▪ procriar() 	<ul style="list-style-type: none"> ▪ Tipo ▪ Raça ▪ Nome ▪ Idade ▪ Sexo ▪ Data de nascimento

Agora, vamos criar outra classe chamada **Ser Humano**. Os métodos e atributos da classe Ser Humano serão:

Ser Humano	
Métodos	Atributos
<ul style="list-style-type: none"> ▪ andar() ▪ alimentar() ▪ dormir() ▪ procriar() ▪ estudar() ▪ trabalhar() ▪ jogar() 	<ul style="list-style-type: none"> ▪ Nome ▪ Etnia ▪ Idade ▪ Sexo ▪ RG ▪ CPF ▪ Data de nascimento

Observe que alguns atributos e métodos são comuns às duas classes.

Animais	
Métodos	Atributos
<ul style="list-style-type: none"> ▪ andar() ▪ alimentar() ▪ dormir() ▪ procurarAlimento() ▪ procriar() 	<ul style="list-style-type: none"> ▪ Tipo ▪ Raça ▪ Nome ▪ Idade ▪ Sexo ▪ Data de nascimento

Ser Humano	
Métodos	Atributos
<ul style="list-style-type: none"> ▪ andar() ▪ alimentar() ▪ dormir() ▪ procriar() ▪ estudar() ▪ trabalhar() ▪ jogar() 	<ul style="list-style-type: none"> ▪ Nome ▪ Etnia ▪ Idade ▪ Sexo ▪ RG ▪ CPF ▪ Data de nascimento

Os métodos duplicados poderão fazer parte de uma classe superior, mais genérica, chamada **Seres Vivos**. Desse modo, os atributos e os métodos comuns que formarão a classe Seres Vivos são:

Seres Vivos	
Métodos	Atributos
<ul style="list-style-type: none"> ▪ <code>andar()</code> ▪ <code>alimentar()</code> ▪ <code>dormir()</code> ▪ <code>procriar()</code> 	<ul style="list-style-type: none"> ▪ Nome ▪ Idade ▪ Sexo ▪ Data de nascimento

Agora, as classes Animais e Ser Humano conterão somente os atributos e métodos específicos delas. Os demais atributos e métodos serão herdados da classe Seres Vivos.

Vamos conhecer os métodos e os atributos específicos de cada classe.

Classe Animais

Atributos: Tipo, Raça.

Método: `procurarAlimento()`

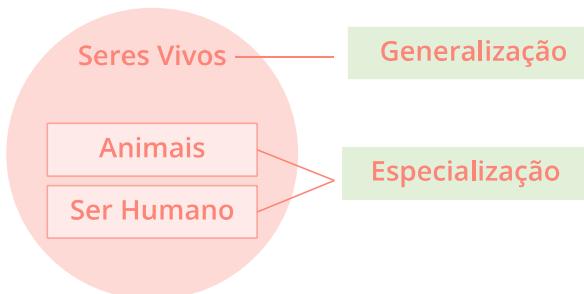
Classe Ser Humano

Atributos: Etnia, RG, CPF.

Método: `estudar()`, `trabalhar()` e `jogar()`

Nesse exemplo, vemos que os atributos e métodos comuns (gerais) estão contidos em uma classe superior chamada Seres Vivos.

A classe Seres Vivos é a generalização dos atributos e dos métodos comuns. Por outro lado, as classes Animais e Ser Humano se tornaram mais específicas. Esse é o conceito da **especialização**.

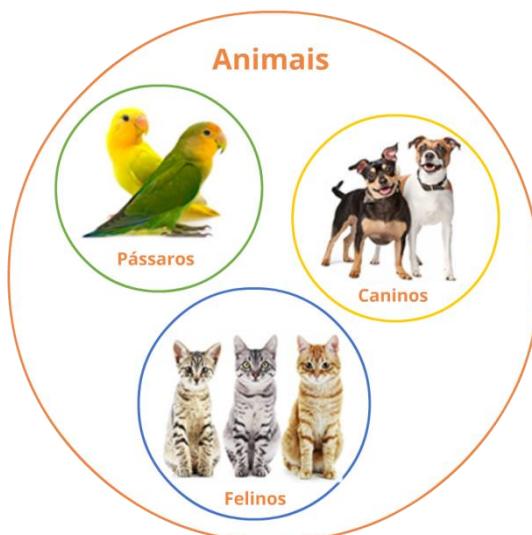


Especialização

Segundo o conceito de especialização, a classe deve ser o mais detalhada possível com relação ao objeto que ela classifica, identificando as diferenças entre os objetos de mesma classe.

No exemplo da classe Animais, suponhamos que queremos separar os pássaros dos felinos e dos caninos. Com isso, estaremos fazendo a especialização da classe Animais.

Observe o esquema.



Cada nova subclasse vai especializar e detalhar mais a classe Animais. Desse modo, serão criadas três novas subclasses – Pássaros, Felinos e Caninos.

Agora, a **classe Pássaros** pode conter o método **voar()**, a **classe Felinos** pode conter o método **unhar()** e a classe **Caninos** pode conter o método **latir()**.

Note que os três métodos citados são específicos de cada objeto descrito. Em outras palavras, tais métodos constituem as diferenças entre os objetos da classe Animais antes da especialização.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre o assunto tratado neste tópico. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Você foi contratado para ser programador de POO sênior em uma empresa de Tecnologia da Informação e precisa implementar, para um jogo de ação, um objeto carro e um objeto avião.

A melhor forma de realizar essa tarefa, com boas práticas de POO, é implementar um objeto:

- Avião, com os atributos: velocidade, combustível, passageiros, portas e autonomia de voo.
- Veículo, com os atributos: velocidade, combustível, passageiros, portas e autonomia de voo.
- Carro, que apresente os atributos: velocidade, combustível, passageiros e portas; e outro objeto Avião, que apresente os atributos: velocidade, combustível, passageiros e autonomia de voo.
- generalizado Veículo, com os atributos: velocidade, combustível e passageiros; e dois objetos especializados: Carro, que herde de Veículos e adicione portas, e Avião, que herde de Veículos e adicione autonomia de voo.

Questão 2

Você é programador OO e precisa programar um objeto Banco que poderá ser instanciado sem a informação do saldo inicial ou com o valor do saldo inicial informado por meio de uma mensagem. Você decidiu fazer uma classe com duas ocorrências do método banco (), sendo que, em uma delas, o método será acionado juntamente com o saldo.

Nesse caso, você optou por utilizar o conceito de:

- herança.
- polimorfismo.
- generalização.
- especialização.

Encerramento do Tópico e do capítulo

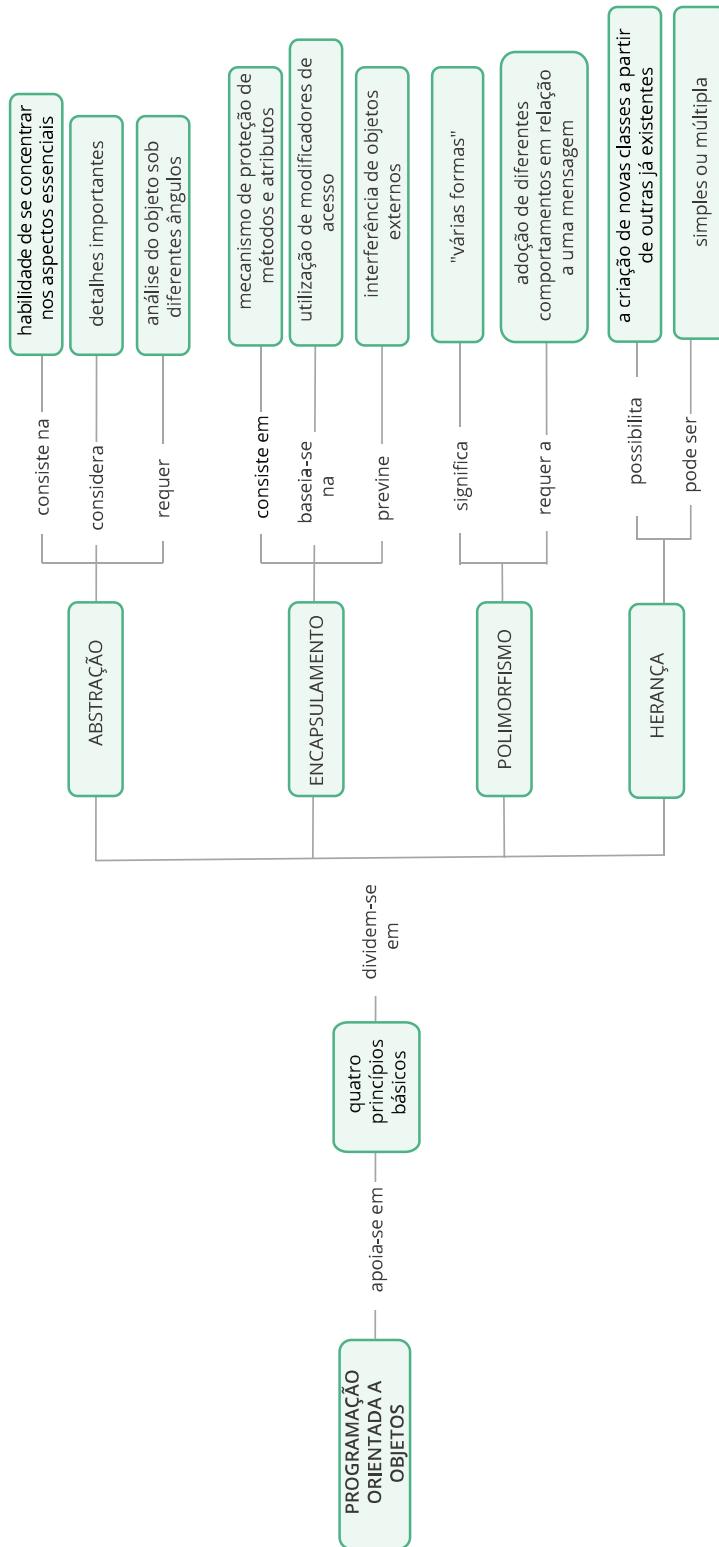
Neste tópico, aprendemos como o polimorfismo reaproveita o código já criado e o transforma em outra proposta. Também analisamos o conceito de herança e suas aplicações. Por fim, vimos como aplicar os conceitos de programação orientada a objetos para escrever códigos.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Você está prestes a concluir o capítulo. No ambiente *on-line*, para facilitar seus estudos, disponibilizamos um PDF com um resumo dos conteúdos abordados neste capítulo, que se encontra na próxima página.

Síntese

Observe, a seguir, o mapa conceitual que sintetiza o conteúdo que acabamos de trabalhar:



CAPÍTULO 3: RELACIONAMENTO ENTRE CLASSES

Tópico 1: Tipos de Relacionamento – Associação, Agregação e Multiplicidade

Neste tópico, vamos entender como a programação orientada a objetos trata o relacionamento entre classes. Para isso, veremos como uma informação pode ser trocada entre as classes e seus objetos, modificando a forma como um objeto se comporta em relação a outro objeto.

Conteúdos:

- Associação
- Agregação
- Multiplicidade.

Ao finalizar este tópico, você será capaz de:

- Identificar o relacionamento entre objetos.
- Compreender como um objeto pode associar-se a outro para buscar uma resposta comum para ambos.

Introdução

Você já reparou que as técnicas da POO estão presentes no funcionamento dos *smartphones*, na forma como procedemos para pagar contas e em outras coisas do cotidiano?

A maioria das atividades comerciais e dos serviços passa pelo mundo da programação. Por esse motivo, interfaces mais fáceis e agradáveis estão sendo desenvolvidas diariamente. Nesse contexto, a programação orientada a objetos é fundamental.

No capítulo anterior, aprendemos que as classes devem interagir entre si para executar uma mensagem ou tarefa comum. Será que essas classes precisam estar ligadas pelo mecanismo da herança? Vamos descobrir!

Associação

As classes também podem se relacionar por meio de associação.

A associação ocorre quando duas classes independentes colaboram entre si, criando um relacionamento entre tais estruturas. Esse relacionamento indica que objetos de uma classe estão relacionados a objetos de outra classe.

Nesse sentido, a associação descreve os vínculos existentes entre as classes que interagem entre si.

Vejamos o esquema a seguir.

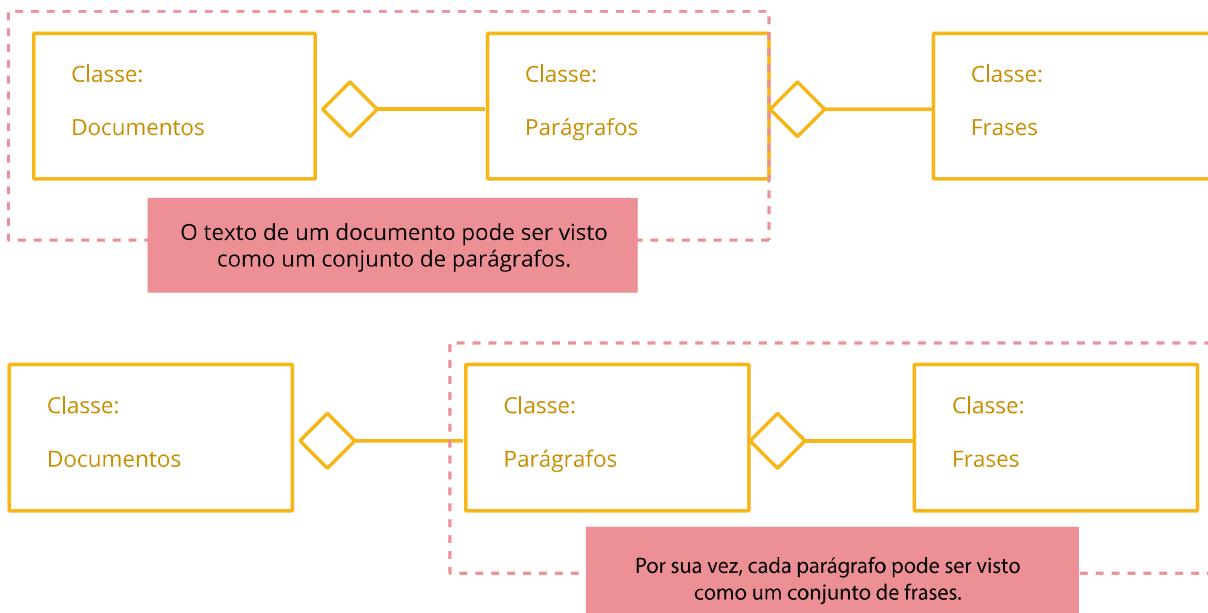


Agregação

Outra forma de relacionar classes é por meio da **agregação**. A agregação é um relacionamento estrutural por meio do qual parte de uma classe é associada a outra classe e partes de objetos são associadas a outras partes de objetos. Com isso, temos uma sequência de montagem.

Na agregação, uma classe se agrupa a outra, mas mantém sua identidade própria. Por conta disso, a agregação é considerada um tipo especial de associação.

A notação de uma agregação é representada por losangos e retas que ligam as classes. Vamos ver um exemplo?



Multiplicidade

A multiplicidade busca estabelecer como as classes que formam uma associação fornecem informações umas às outras. Além disso, a multiplicidade permite especificar o nível de dependência entre as classes envolvidas.

Normalmente, a multiplicidade descreve os valores mínimos e máximos que uma associação pode conter. Tal informação aparece da seguinte forma nos diagramas:

Multiplicidade 0,1 – no mínimo, **zero** e, no máximo, **um**.

Indica que um objeto pode ou não estar relacionado com outros objetos de uma outra classe.

Classe com classe – objeto com objetos de mesma classe.

Multiplicidade 1,1 – **um** e **somente um**.

Indica que um objeto se relaciona com outros objetos de uma outra classe.

Classe com classe – objeto com objetos de mesma classe.

Multiplicidade 0,n – no mínimo, **nenhum** e, no máximo, **muitos**.

Indica que pode haver ou não relacionamentos. Quando houver relacionamentos, eles podem ser muitos, entre diversas classes e diversos objetos.

Multiplicidade n – **muitos**.

Indica que muitos objetos estão envolvidos nos relacionamentos.

Multiplicidade 1,n – no mínimo, **um** e, no máximo, **muitos**.

Indica que existe um relacionamento entre os objetos pelo menos. No entanto, pode haver muitos relacionamentos entre os objetos e suas classes.

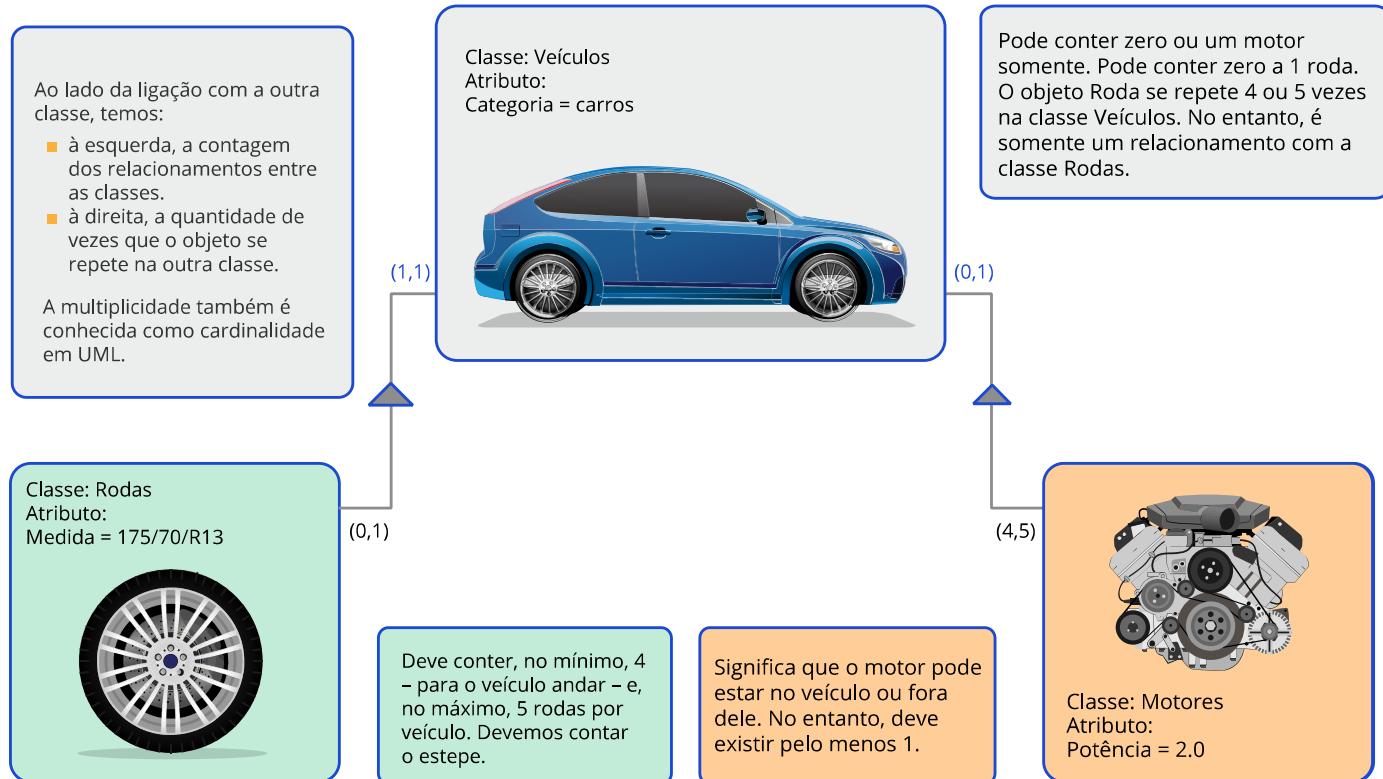
Multiplicidade 2,5 – no mínimo, **dois** e, no máximo, **cinco**.

Indica que existem dois relacionamentos entre os objetos pelo menos. Os relacionamentos podem chegar a cinco, não mais do que isso.



A notação da multiplicidade indica os números de mínimo e máximo entre parênteses, ao lado da linha de relacionamento.

Agora, vejamos, no infográfico a seguir, um exemplo de representação de multiplicidade.

REPRESENTAÇÃO DE MULTIPLICIDADE**Relembrando Conceitos****Vídeo**

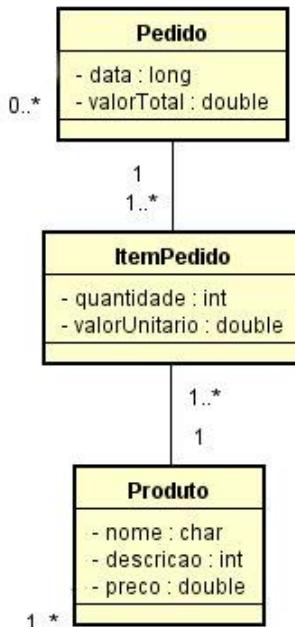
Vamos aprender um pouco mais? No ambiente *on-line*, você pode assistir ao audiovisual **Relacionamento entre Classes** para rever alguns conceitos considerados neste capítulo.

Exercícios de Fixação

Agora, veja o quanto você sabe sobre o assunto tratado neste tópico. Realize os exercícios a seguir e aproveite para fixar melhor os conceitos vistos até aqui.

Questão 1

Uma multinacional está desenvolvendo um *software* para uma revenda autorizada de automóveis. Como programador, você foi encarregado de validar, junto com o analista de sistemas, o diagrama de classes a seguir:



Analisando o diagrama, marque **V** para verdadeiro e **F** para falso em relação às afirmações a seguir.

informações	V	F
Pedido, ItemPedido e Produto são os nomes das classes.	<input type="radio"/>	<input type="radio"/>
Quantidade e valor unitário são métodos da classe ItemPedido.	<input type="radio"/>	<input type="radio"/>
A identificação 1 e * representam a cardinalidade dos relacionamentos.	<input type="radio"/>	<input type="radio"/>
A relação entre Pedido e ItemPedido poderia ser representada por um relacionamento do tipo Agregação.	<input type="radio"/>	<input type="radio"/>

Questão 2

Um cliente solicitou uma consulta ao departamento de TI a respeito dos motivos de a programação orientada a objetos ser considerada mais eficiente do que programação tradicional.

Dessa forma, marque **V** para os argumentos verdadeiros e **F** para os argumentos falsos:

argumentos	V	F
Na POO, os dados são identificados pelos seus métodos, e os processos são identificados por meio dos atributos das classes.	<input type="radio"/>	<input type="radio"/>
A POO permite que novos códigos de programação sejam acrescidos ou excluídos com igual facilidade, além de evitar códigos sem função dentro do sistema.	<input type="radio"/>	<input type="radio"/>
A programação orientada a objetos pode ser dividida facilmente entre diversos programadores, permitindo o desenvolvimento simultâneo para a entrega mais rápida do sistema.	<input type="radio"/>	<input type="radio"/>
A análise e programação tradicional (procedural) permite uma melhor representação do mundo real e, por isso, é utilizada em grande escala no desenvolvimento de sistemas.	<input type="radio"/>	<input type="radio"/>

Encerramento do Tópico

Neste tópico, aprendemos a identificar o relacionamento entre objetos.

Vimos como uma informação pode ser trocada entre as classes e seus objetos, modificando a forma como um objeto se comporta em relação a outro objeto.

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Vamos prosseguir em nossos estudos? Siga para o próximo tópico!

Tópico 2: Vantagens da Programação Orientada a Objetos

Neste tópico, veremos as vantagens da programação orientada a objetos. Além disso, vamos conhecer algumas dicas para modelagem de um sistema orientado a objetos.

Conteúdos:

- Vantagens da POO
- Dicas para a modelagem de um sistema orientado a objetos.

Ao finalizar este tópico, você será capaz de:

- Compreender a importância da orientação a objetos e preparar-se para realizar a modelagem de um sistema.

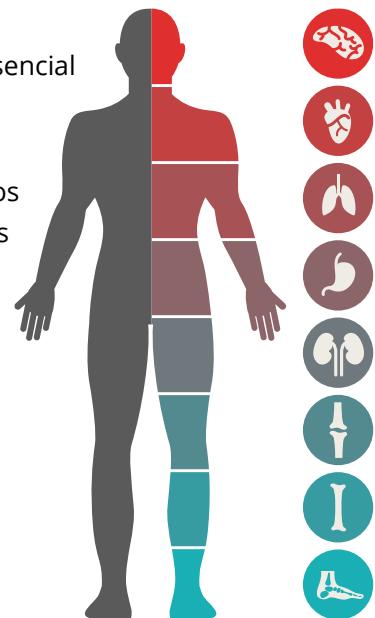
Vantagens da Programação Orientada a Objetos (POO)

A programação orientada a objetos (POO) oferece muitas vantagens em relação ao método tradicional linear e sequencial.

A POO concebe o sistema como um organismo, em que cada órgão é essencial e todos os órgãos interagem entre si.

Vamos considerar o organismo humano. Como órgão, o coração é menos importante do que o cérebro? Podemos sobreviver sem o fígado? Sem as pernas, podemos locomover? Quem comanda suas ações pode ficar sem os braços e as pernas?

Como podemos notar, tudo está interligado e interage a todo momento. Esse enfoque pode ser aplicado tanto na análise do sistema quanto na programação propriamente dita.



Vamos conhecer as principais vantagens da POO?

Unificação de dados e processos

Os dados são identificados por meio dos atributos das classes. Já os processos são identificados pelos métodos. A interação entre os objetos é definida pelo relacionamento entre os objetos e suas classes.

Com isso, nesse modelo de programação, existe total coerência entre dados e processos do mundo real, e atributos e métodos do mundo da programação.

Consistência entre análise e desenvolvimento

A análise tradicional busca mapear, isoladamente, as estruturas de dados e os processos que tratam e alteram tais dados. Normalmente, isso é feito com ferramentas como o Modelo de Entidades e Relacionamento (MER ou ER) e o Diagrama de Fluxo de Dados (DFD).

A análise tradicional é um tipo de modelagem exclusivamente conceitual. Não existe relação direta entre as entidades conceituais e as entidades físicas a serem implementadas. Com isso, é praticamente impossível identificar o processo de DFD que está sendo implementado em um programa qualquer – a não ser que isso esteja claramente indicado.

Na POO, isso já está implícito. As classes, os objetos e seus relacionamentos definidos são exatamente os mesmos que serão implementados.

Reutilização do código já implementado

Uma vez que já temos os métodos e os atributos declarados e funcionando, podemos criar novas classes que herdem essas funcionalidades. Em outras palavras, podemos atribuir novas funções a partir das funções já existentes, sem precisar modificar nada do que já está pronto – herança, polimorfismo e associação.

Também podemos utilizar bibliotecas já desenvolvidas e adicioná-las em nosso sistema. Com isso, reduzimos o custo de desenvolvimento de um novo sistema consideravelmente.

Multidesenvolvimento

Os desenvolvedores podem programar partes do sistema a ser desenvolvido simultaneamente. Isso pode ser feito a partir de um conjunto de bibliotecas disponíveis de comum acordo. Cada uma dessas partes será acrescida ao projeto final do sistema.

A especificação de uso pode ser codificada por cada desenvolvedor separadamente. No entanto, é preciso utilizar os mesmos objetos e classes já definidos e desenvolvidos anteriormente, de comum acordo. O desenvolvedor apenas necessita conhecer as classes e os objetos desenvolvidos pelo outro desenvolvedor, bem como familiarizar-se com eles, a fim de utilizá-los em seus processos.

Facilidade de manutenção

Quando necessitamos modificar ou corrigir uma classe, estamos dizendo que é possível identificar qual classe está com problemas em seus métodos e atributos. Isso também indica que podemos criar novas classes, métodos, atributos e relacionamentos facilmente, para atender as novas demandas do sistema.

Tal possibilidade permite que novos códigos de programação sejam acrescidos ou excluídos com igual facilidade. Da mesma forma, não teremos códigos sem função dentro do sistema.

Modelagem de um Sistema Orientado a Objetos

A modelagem de um sistema baseado na orientação a objetos é um meio de descrever os sistemas e as necessidades do mundo real.

Na programação, um objeto representa a abstração de uma entidade do mundo real, de modo que informação e comportamento estão ligados. Nesse sentido, informações e comportamentos são traduzidos para a programação como atributos e métodos.

A grande vantagem disso está no fato de que o objeto definido na fase de análise mantém as mesmas características desde o programador até o usuário final.



Uma vez estabelecidas as classes e os objetos, vamos ajustando as classes e seus relacionamentos, melhorando o modelo. Esse é um processo natural e previsto.

Como podemos nos preparar para o desenvolvimento de um modelo de orientação a objetos? Observe os principais passos.

Passo 1

Não inicie um modelo de objetos apenas definindo classes, atributos, métodos, associações e heranças. A primeira coisa a fazer é compreender o contexto do problema a ser resolvido.

Passo 2

Tente manter o modelo simples. Evite complicações desnecessárias. Se necessário, faça uma decomposição do problema em vários problemas simples.

Passo 3

Escolha os nomes com critério e cuidado. Os nomes corretos são importantes e carregam conotações poderosas. Outros desenvolvedores terão dificuldades se o nome da classe não for condizente com o que é executado. Os nomes devem ser descritivos, claros e não deixar margem a dúvidas.

Passo 4

Identifique os métodos e os atributos essenciais. Não coloque identificadores de objetos dentro de outros objetos como atributos. Modele os identificadores com clareza, indicando-os como associações. Isso torna o modelo mais claro e independente da implementação.

Passo 5

Identifique os relacionamentos necessários entre os objetos. Evite associações que envolvam muitas classes de objetos. Na maioria das vezes, tais associações podem ser decompostas em associações simples, tornando o modelo mais claro. É muito mais fácil descrever vários relacionamentos simples do que um relacionamento entre várias classes.

Passo 6

Tente evitar hierarquias de generalização muito densas ou heranças complexas.

Passo 7

Sempre documente seus modelos de objetos, classes e relacionamentos. A UML – *Unified Modeling Language* – é perfeita para isso.

Passo 8

A complexidade e o detalhamento de classes e objetos têm um limite. O limite está ligado a seu uso prático. Não adianta definir muitos atributos e métodos se vamos utilizar somente alguns. A quantidade de atributos e métodos depende da complexidade do problema.

Relembrando Conceitos

**Vídeo**

Vamos aprender um pouco mais? No ambiente *on-line*, você pode assistir ao audiovisual **Programação Orientada a Objetos - POO** para rever e fixar melhor alguns conceitos vistos até aqui.

Ética – Pense Nisso!

Estamos chegando ao final deste conteúdo. Antes, no entanto, é importante que façamos uma pausa para refletir sobre um assunto muito importante: a ética. Preparado?

Em breve, você vai tornar-se um desenvolvedor de sistemas. Desse modo, com certeza, vai fazer consultas frequentes à *web* para tirar dúvidas ou aprender a utilizar uma classe, uma biblioteca ou um método.

Nessas ocasiões, antes de copiar e colar códigos escritos por terceiros, verifique se o autor autorizou o uso por qualquer pessoa ou empresa, sem restrições.

É muito importante que a licença seja explícita. Caso contrário, você e a empresa em que trabalha podem sofrer processos caríssimos de uso indevido de direitos autorais.

Atualmente, existem *softwares* e bancos de dados que fazem uma busca completa em seu *software* para ver se seu código não está infringindo leis de patentes ou de direitos autorais.

Normalmente, empresas que compram serviços verificam se existe código escrito sem permissão de uso. Por conta disso, não use o código se tiver dúvida sobre sua licença de uso.

Muitos desenvolvedores autorizam somente a cópia de seu código para uso individual. Desse modo, quando o uso se torna comercial, devemos pagar pelos direitos de uso.

Pesquise os tipos de licença mais comuns de código aberto e suas restrições. Siglas como GNU e GPL serão parte de seu mundo de desenvolvedor.

Jamais utilize componentes prontos – como DLLs ou executáveis –, mesmo que sejam gratuitos.

Muitos *hackers* são desenvolvedores normais, mas fazem códigos maliciosos e distribuem-nos pela *web* gratuitamente.

Tais códigos podem abrir portas em *firewalls*, replicar o que é digitado no teclado, capturar senhas de usuários da rede de empresas, entre outras coisas. Há uma infinidade de códigos que podem fragilizar a segurança de seu ambiente de desenvolvimento.

Se, ainda assim, valer a pena assumir os riscos, prefira componentes com código-fonte distribuído pelo autor para ser averiguado.

Dê preferência a *sites* oficiais de grandes empresas de *software*. Muitas empresas liberam grande quantidade de informações, e linhas de código prontos e com exemplos.

Normalmente, essas licenças de uso são gratuitas. No entanto, não há garantias de que vão funcionar adequadamente.

Dessa forma, o risco é todo seu!

Como desenvolvedor, autor e produtor de *softwares* ou soluções, você deve assumir uma postura mais séria. Você não vai querer que sua ideia seja copiada e reproduzida sem autorização, não é mesmo?

A linha do que é ético ou não é bem tênue. No papel, pode parecer "preto no branco", mas é bem conflitante na vida real.

Por exemplo, você faz *download* de filmes e músicas pela internet gratuitamente? Isso parece ético com os autores, os cantores e os produtores do produto que você consome? Utilizar trabalhos de outros e dizer que foi você quem desenvolveu é ético?

Nos países de primeiro mundo, a corrupção, o plágio, a cópia ilegal e a infração de direitos autorais têm severas implicações legais e econômicas.

Antes de ser lei, ética é atitude. Ética é exercício contínuo de caráter. Não esqueça disso!



Saiba mais!

No Brasil, os princípios, as garantias, os direitos e os deveres para o uso da Internet são tratados pela Lei 12.965/14, conhecida como o Marco Civil da Internet.

Para ler a Lei na íntegra, acesse:

http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm

Exercícios de Fixação

Questão 1

Para um trabalho em grupo, do curso de programação, você foi encarregado de apresentar as características e as vantagens da POO.

Assim, você iniciou sua apresentação explicando que a modelagem de um sistema baseado em orientação a objetos é um meio de descrever os sistemas e as necessidades do mundo real.

A seguir, você esclareceu que:

- uma vez estabelecidos os relacionamentos, ajustamos as classes e seus objetos.
- um objeto representa a modelagem de uma informação ou comportamento.
- Informação e comportamento são traduzidos para a programação como abstrações, classes e polimorfismo.
- o objeto definido na fase de análise mantém as mesmas características desde o programador até o usuário final.

Questão 2

Como desenvolvedor de sistemas, você precisou fazer consultas na *web* para tirar dúvidas sobre uma biblioteca, durante o atendimento a um cliente.

Nessa ocasião, para agir guiado pela ética, você optou por:

- recorrer, especificamente, a códigos autorizados para uso individual.
- verificar se o autor autorizou o uso irrestrito para pessoas ou empresas.
- utilizar apenas códigos que não apresentavam licença explícita, para evitar problemas.
- copiar e colar códigos que precisou utilizar, uma vez que estavam disponíveis na rede.

Encerramento do Tópico e do capítulo

Neste tópico, conhecemos as vantagens da programação orientada a objetos e você aprendeu algumas dicas para iniciar a modelagem de sistema orientado a objetos.

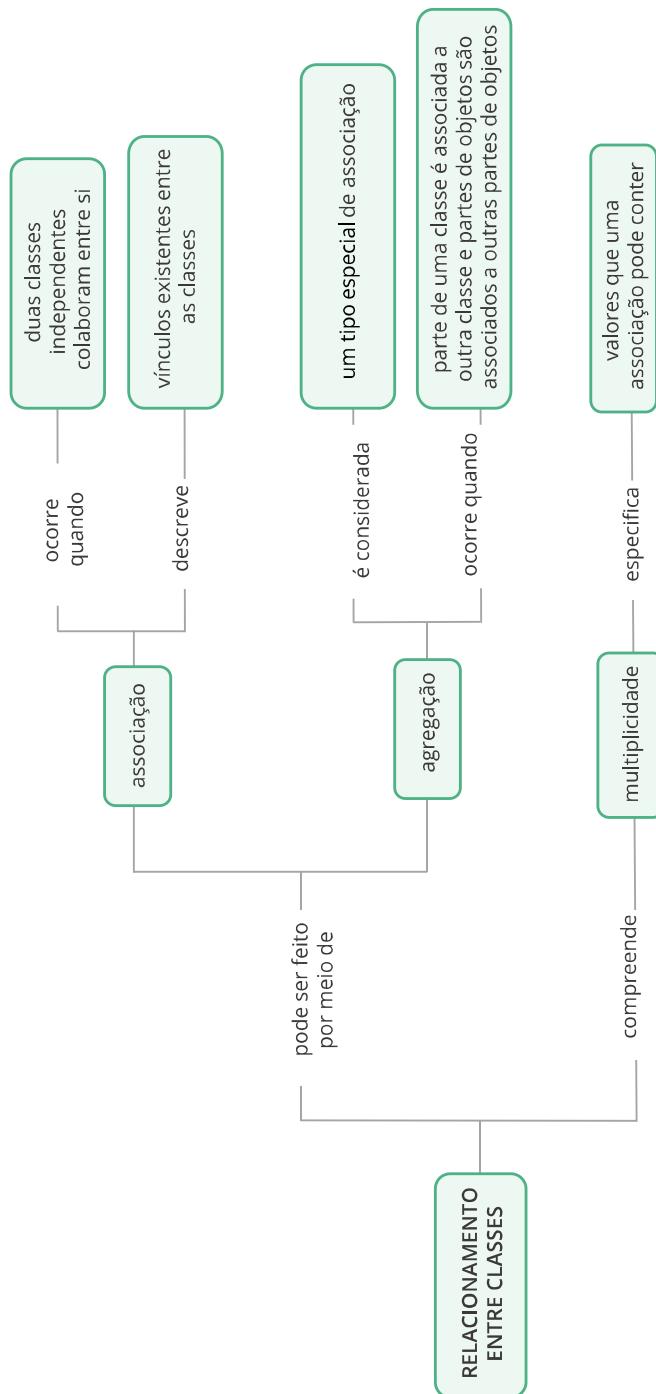
Agora, você compreende a importância da orientação a objetos e está preparado para realizar a modelagem de um sistema!

Caso ainda tenha dúvidas, você pode voltar e rever o conteúdo.

Você está prestes a concluir o capítulo. No ambiente *on-line*, para facilitar seus estudos, disponibilizamos um PDF com um resumo dos conteúdos abordados neste capítulo, que se encontra na próxima página.

Síntese

Observe, a seguir, o mapa conceitual que sintetiza o conteúdo que acabamos de trabalhar:



LEITURA RECOMENDADA

Amplie seu conhecimento com algumas leituras.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. *Análise orientada a objetos*. 2. ed. Florianópolis: Visual Books, 2006. p. 105-108.

BARNES, David J.; KOLLING, Michael. *Programação orientada a objetos com Java*. 4. ed. São Paulo: Prentice Hall, 2009. cap. 5.

REFERÊNCIAS BIBLIOGRÁFICAS

Apostila de orientação a Objetos com Java.

BARNES, David J.; KOLLING, Michael. *Programação orientada a objetos com Java*. 4. ed. São Paulo: Prentice Hall – Br, 2009.

CAMARA, Fabio. *Orientação a objeto com .NET*. 2. ed. Florianópolis: Visual Books, 2006.

CARVALHO, Adelaide. *Práticas de C# – Programação orientada por objetos*. Lisboa: FCA, 2011.

Conceitos Básicos de Programação Orientada a Objetos

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. *Análise orientada a objetos*. 2. ed. Florianópolis: Visual Books, 2006.

INTRODUÇÃO A PROGRAMAÇÃO ORIENTADA A OBJETOS COM VISUAL BASIC.NET

SHUTTERSTOCK. Disponível em: <https://www.shutterstock.com>. Acesso em: dez. 2016.

ENCERRAMENTO

Parabéns!

Chegamos ao final deste conteúdo! Esperamos que você tenha gostado das técnicas apresentadas de Programação Orientada a Objetos (POO).

Sem dúvida, os conceitos serão mais bem-entendidos quando você iniciar a programação propriamente dita.

Você verá que a POO vai facilitar muito suas tarefas. Principalmente, em relação à manutenção, às melhorias e ao reuso de código já escrito.

Lembre-se de que, para ser um profissional de destaque, você precisa estar sempre se aperfeiçoando. Então não pare por aqui, mantenha-se atualizado e faça de todas as suas experiências profissionais um caso de sucesso.

<https://fundacao.bradesco/>