UNIP
UNIVERSIDADE PAULISTA
Interativa

Programação Orientada a Objetos I

Autores: Prof. Helder Frederico Lopes
Prof. Olavo T. Ito
Colaboradores: Prof. Luciano Soares Souza
Profa. Ana Carolina Bueno Borges
Prof. Gley Fabiano Xavier

Professores conteudistas: Helder Frederico Lopes / Olavo T. Ito

Helder Frederico Lopes

É mestre em Ciências pela Faculdade de Medicina da USP – Departamento de Neurologia, técnico em Processamento de Dados e tecnólogo em Tecnologia em Gestão de Sistemas de Informação pela Universidade Nove de Julho.

Possui 20 anos de experiência na área de Ciência da Computação e Informática, com ênfase em desenvolvimento de *software*, em sua empresa e também como consultor.

É um pesquisador atuante em áreas ligadas à inteligência artificial, lógica paraconsistente e redes neurais artificiais.

Atualmente, atua como docente, empresário e consultor de treinamentos e implantação de processos para transformação de cultura em equipes e profissionais.

Olavo T. Ito

É mestre em Engenharia de Produção pela UNIP, bacharel em Física pelo Instituto de Física da USP e licenciado em Ciências pela Faculdade de Educação da USP. Possui curso completo de Língua e Literatura Japonesa pela FFLCH – USP.

Seus primeiros passos no mundo da programação de computadores foram dados com calculadoras programáveis HP25C e Sharp Pocket Computer. Nessa mesma época, na faculdade, estudou lógica de programação, perfurando cartões para praticar o aprendizado em computador de grande porte, o Burroughs B6700.

Durante a iniciação científica no Departamento de Física Experimental do Instituto de Física, mergulhou no mundo da programação usando a linguagem Fortran em um dos dois únicos computadores IBM 360 remanescentes no mundo na época, pertencente ao laboratório do acelerador de partículas Pelletron, desenvolvendo programas de ajustes estatísticos. Nesse período, início dos anos 1980, quando começaram a ser fabricados os primeiros microcomputadores no Brasil, seu grupo de pesquisa recebeu a doação de um Itautec I-7000 que tinha tanto poder de processamento quanto o gigantesco IBM360, além de permitir o uso de outras linguagens de programação além do Fortran, tendo à sua disposição o Basic, o xBase, o C, o Pascal, o Algol, o Cobol e muitos outros.

A experiência com programação o levou a sair da vida acadêmica para trabalhar na iniciativa privada, mas sem deixar de lado o compromisso com o ensino. Trabalhou em banco, consulado, *holdings*, indústria e comércio, especializando-se em implantação de ERP (*Enterprise Resource Planning* – Sistemas de Gestão Empresarial).

É professor dos cursos de graduação de Sistemas de Informação e Ciências da Computação e do curso superior tecnológico de Análise de Desenvolvimento de Sistemas.

Dados Internacionais de Catalogação na Publicação (CIP)

L864p Lopes, Helder Frederico.

Programação orientada a objetos I. / Helder Frederico Lopes, Olavo T. Ito. – São Paulo: Editora Sol, 2015.
184 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ano XVII, n. 2-026/15, ISSN 1517-9230.

1. Linguagem de programação. 2. Integração entre classes. 3. Interação entre objetos. I. Lopes, Helder Frederico. II. Ito, Olavo T. III Título.

CDU 681.3

U500.71 – 19

Sumário

Programação Orientada a Objetos I

Unidade IV

APRESENTAÇÃO

Este livro-texto tem como objetivo roteirizar o processo de aprendizagem do aluno, tal qual um guia.

Todo o conteúdo aqui presente foi elaborado de maneira a situar o aluno no conteúdo teórico por meio de exemplos e desafios que permitam a ele visualizar seu progresso e compreender suas dúvidas. Tal metodologia consiste na divisão desse material em unidades ordenadas por grau de aprendizado, ou seja, para compreender a última unidade, o aluno necessita passar por todas as outras, de maneira a acumular o conhecimento necessário para a compreensão da disciplina. Porém, cada unidade abordará os assuntos de forma independente. Desse modo, o aluno com mais facilidade de aprendizado (ou que tenha uma carga de conhecimento prévio) poderá avançar seus estudos sem prejuízo ao conteúdo, enquanto aquele que está iniciando seus passos tem a garantia de passar pelas etapas corretas para a construção do seu conhecimento.

Inicialmente, veremos os conceitos básicos da *Programação Orientada a Objetos*, ou seja, sua história, as vantagens sobre a programação procedural e seus fundamentos.

Em seguida, será introduzida a linguagem de programação C# da plataforma Microsoft .NET, com a qual serão exemplificados com aplicações práticas os principais conceitos da programação orientada a objetos.

Avançando nossos estudos, abordaremos cada um dos fundamentos da programação orientada a objetos. A compreensão de tais fundamentos é essencial para o domínio desta disciplina. É recomendável (mas não impeditivo) que o aluno siga adiante apenas quando compreender bem tais fundamentos.

Encerraremos aprofundando os fundamentos apresentados anteriormente, abordando conceitos mais avançados que permitem o uso prático da programação orientada a objetos em sistemas computacionais.

Bom estudo!

INTRODUÇÃO

A interação dos seres humanos, que podemos chamar genericamente de "homens", com os seres digitais, que genericamente podemos chamar de "computadores", passou por uma série de melhorias. No início, somente poucos "homens" conseguiam fazer essa interação, pois elas consistiam em chaves que tinham luzinhas que acendiam ou apagavam e em cartões cheios de buracos, que exigiam um grande conhecimento. Algumas décadas depois, qualquer "homem" consegue utilizar um "computador" usando a ponta do dedo e tocando numa chapa de vidro – e, porque não, simplesmente olhando e piscando – em uma imagem virtual.

Para pensar em como o "homem" interage com o "computador", há a figura do "programador", aquele que tem a incumbência de fazer o "computador" trabalhar auxiliando o "homem" a calcular, a se informar, a se divertir.

Ao longo do tempo, a maneira de pensar como programar o computador também mudou. Antes, era um processo linear e único. Uma vez iniciado o programa, o "homem" que o fosse utilizar tinha que seguir um roteiro criado pelo programador com poucas variantes. Atualmente, ao executar um programa, o "homem" pode decidir qual o roteiro vai seguir. No meio de uma tela de cadastro, ele pode apertar um botão e cancelar tudo; enquanto faz uma coisa, ele pode decidir usar outra coisa, bastando tocar em um desenho. Dessa forma, o programador precisou mudar a maneira de pensar um programa, precisou aprender e entender um novo modelo de construção sem deixar de lado a experiência do modelo já conhecido.

A maneira antiga de pensar se chama programação estruturada (ou funcional, ou procedural); já a nova maneira de pensar um programa se chama programação orientada a objeto. Afinal, qual é a maneira de pensar utilizando esse novo modelo, que é a programação orientada a objeto? A resposta veremos neste livro-texto.

Unidade I

CONHECENDO A PROGRAMAÇÃO ORIENTADA A OBJETOS

Nesta unidade apresentaremos a história e os conceitos fundamentais da programação orientada a objetos.

Ao final desta unidade, o aluno deverá ser capaz de saber:

- os motivos que levaram à criação de um novo conceito de desenvolvimento de aplicações computacionais: a programação orientada a objetos;
- as vantagens em programar com esta metodologia;
- os fundamentos que regem a programação orientada a objetos.

1 A PROGRAMAÇÃO ORIENTADA A OBJETOS

1.1 Histórico da programação orientada a objetos

Ao contrário do que muita gente pensa, a programação orientada a objetos (POO) não é uma inovação tão recente.

Em 1963 Ivan Sutherland, do MIT, desenvolveu um editor gráfico chamado Sketchpad no seu curso de doutorado. O Sketchpad foi o primeiro editor gráfico orientado a objetos, pois além de colocar pontos coloridos na tela, possibilitava a criação de objetos que poderiam ser manipulados em outros projetos. Uma das suas características mais importantes era a possibilidade de definir um desenho mestre a partir do qual seriam criados os desenhos instanciados, sendo cada uma dessas instâncias semelhantes a ele. Se o desenho mestre fosse alterado, todas as instâncias seriam alteradas da mesma forma.

As ideias do Sketchpad foram os pontos de partida para os conceitos herança em orientação a objetos.

A programação em si, usando os conceitos de orientação a objetos, também teve origem na década de 1960, na Noruega, no Centro Norueguês de Computação, pelos pesquisadores Kristen Nygaard e Ole-Johan Dahl. Nessa época, foram criados os conceitos sobre classe e herança (que serão apresentados em detalhes) implementados na linguagem Simula 67 (LINDEN, 2008; SEBESTA, 2011).

Saiba mais

Para saber mais sobre a história dos pesquisadores do Simula 67, acesse o *site* do Centro Norueguês de Computação:

<http://www.nr.no/en/about-main>.

Outro importante pesquisador que contribuiu para o desenvolvimento da POO foi Alan Curtis Kay, na Xerox, nos EUA. Em seus experimentos, Alan Kay estudava maneiras de interpretar os problemas do mundo real de uma maneira que o ser humano conseguisse abstrair os conceitos fundamentais de tal problema e inferi-los no mundo computacional. Dessa forma, Alan Kay percebeu que um substantivo isolado na mente de uma pessoa gera uma forma concreta, mas um verbo isolado na mente de uma pessoa não. Logo, um verbo faz parte de um substantivo.

Partindo dessa premissa, torna-se fácil para nós, seres humanos, perceber que "correr", isoladamente, não gera uma informação real (torna-se uma ação sem sentido). No entanto, "crianças correm" gera uma informa real em nossa mente, na qual o substantivo é "crianças" e o verbo é "correr".

A partir dessas observações, Alan Kay definiu os princípios da POO:

- qualquer coisa no mundo real é um objeto;
- objetos realizam tarefas por meio de ações;
- cada objeto pode ser agrupado em tipos (classes);
- um tipo de objeto (classe) deve agrupar objetos por similaridade de forma e comportamento;
- cada tipo de objeto (classe) é organizado hierarquicamente.

As técnicas de POO são cognitivamente válidas, visto que a utilização dessas técnicas segue a maneira natural que o ser humano abstrai (imagina) o mundo real. Portanto, a POO é uma metodologia de desenvolvimento de *software* em que a principal vantagem é a proximidade com a forma que os seres humanos visualizam e entendem o mundo ao seu redor, tendo como principal objetivo facilitar a modelagem e o desenvolvimento de sistemas por meio da interação entre objetos.

1.2 Vantagens e objetivos da OO

Desde sua criação até os dias de hoje, muitas linguagens foram criadas e gradativamente incorporando os conceitos da POO por completo.

A POO pode ser considerada uma evolução da programação estrutural/modelar, haja vista a popularização dos computadores em diversos ambientes com o passar do tempo, ampliando assim o leque de aplicações. Consequentemente, isso aumentou também a demanda na indústria de *software*.

Diferentemente da programação estrutural/modular, na qual temos uma sequência de instruções com interação com o usuário, chamada a sub-rotinas e tomadas de decisão, a POO vai além desse modelo de programação.

Na programação estrutural/modelar, caso o programador queira reutilizar o código de um programa em outro, é feita literalmente uma cópia desse trecho de código para o novo programa. Não há nada de errado em reutilizar códigos; porém, caso o programador tenha reutilizado seu código em diversos programas e alguma especificação for alterada justamente no trecho de código que ele reutilizou, o programador terá que realizar a atualização em todos os programas nos quais ele faz o reuso de código. Com esse exemplo, fica claro que a manutenção em programas torna-se cada vez mais custoso, pois o programador utilizará muitas horas de trabalho para atualizar as aplicações necessárias.

Com a alta demanda na indústria de *software*, as empresas procuram maneiras de minimizar custos e otimizar a produtividade dos programadores. Nesse aspecto, a POO se apresenta como uma ferramenta que auxilia o objetivo de modelagem de sistemas de maneira natural e alta reusabilidade de código, ou seja, o programador modela a solução e escreve o código apenas uma vez, mas utiliza a solução modelada com diversas outras aplicações.

Outro exemplo de vantagem da POO está justamente em sua modelagem de soluções. Por ser uma modelagem com base em abstração do ambiente real, sua estrutura se apresenta com muita facilidade a outros programadores e, portanto, a quantidade de documentação gerada fora da aplicação tende a diminuir, sendo quase autoexplicativo. Podemos ter um breve comparativo entre as vantagens e desvantagens do uso da POO no quadro a seguir.

Observação

Vale ressaltar aqui que a POO não isenta o programador a não gerar documentação. O que ocorre é que a documentação gerada pela equipe de desenvolvimento foca mais na regra de negócio do que na explicação da estrutura do programa.

A princípio, não há muita diferença entre a POO e a programação estrutural/modular, mas analisando seus conceitos fundamentais fica claro o quão poderosa é essa técnica de programação.

Quadro 1 – Vantagens e desvantagens de um sistema OO

Vantagens	Desvantagens
• Os sistemas POO geralmente possuem uma distribuição de código um pouco mais lógica, funcional e melhor encapsulada, tornando a manutenção e a extensão do código mais fácil e com menos riscos de inserção de erros. • É mais fácil reaproveitar o código. • É mais fácil gerenciar o desenvolvimento deste tipo de *software* em grandes equipes, partindo de uma especificação UML antes de iniciar o desenvolvimento do *software* em si.	• O aprendizado do paradigma de programação orientada a objetos é bem mais complicado no início do que os sistemas estruturados. • Para começar a programar, é necessário ter uma base conceitual bem-formada, ao contrário da programação procedural tradicional, em que basta decorar meia dúzia de comandos para que seja possível fazer programa simples. • Em termos de velocidade de processamento, dificilmente uma linguagem orientada a objetos conseguirá ter um desempenho superior a linguagens não orientadas a objetos.

Segundo a Microsoft, a programação orientada a objeto (POO) é apoiada em três pilares: encapsulamento, herança e polimorfismo (MICROSOFT, 2014; SINTES, 2002).

Observação

É importante ressaltar que a sintaxe e a semântica da linguagem de programação permanecem inalterados. Se uma pessoa que conhece apenas o modelo estruturado vir sem muita atenção a listagem de um programa OO, ela achará que se trata de um programa comum com alguns poucos acréscimos na linguagem de programação.

A programação orientada a objetos apresenta uma nova forma de pensar a construção de um programa.

2 PRINCIPAIS CONCEITOS

A POO possui alguns conceitos importantes que devem ser estudados, compreendidos e exercitados para que o aluno consiga trabalhar, de forma correta, no desenvolvimento de *softwares*.

A melhor forma de visualizar projetos de *softwares* desenvolvidos por POO se dá a partir da utilização de gráficos, mais especificamente a *Unified Modeling Language* (UML). Portanto, os gráficos utilizados neste material seguirão as definições da UML, que é um padrão no que diz respeito a projetos de *softwares* em POO.

Saiba mais

Para entender mais sobre UML, leia a obra a seguir:

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML*: guia do usuário. 2 ed. Rio de Janeiro: Campus , 2006.

Programar orientado a objetos deve ser uma tarefa natural, desde que o aluno aprenda a pensar orientado a objeto. Quando crianças, as pessoas desenvolvem uma orientação a objetos natural que, com o passar do tempo, vai se perdendo, dando lugar a um pensamento procedural e estruturado.

Saiba mais

Existem vários programas de edição de diagramas UML, alguns deles gratuitos. Você pode encontrar a versão gratuita do Visual Paradigm, o Visual Paradigm Community Edition, no *site* a seguir:

DOWNLOAD Visual Paradigm Community Edition. [s.d.]. Disponível em: <http://www.visual-paradigm.com/download/community.jsp>. Acesso em: 27 out. 2014.

Vejamos o seguinte exemplo: para se dirigir um carro, nós simplesmente nos sentamos, ajustamos os controles (tais como banco, altura do volante, espelhos e farol), damos partida e dirigimos! Então, podemos dizer que nós somos usuários de um sistema chamado Carro. Vamos analisá-lo:

- Este Carro é composto por uma série de peças que chamamos de **classe**.
- Tanto o Carro como essas classes possuem uma série de características próprias. Para essas características, damos o nome de **atributos**.
- Cada classe, assim como o Carro, também possui funções (comportamentos) específicas. Essas funções são chamadas de **métodos**.

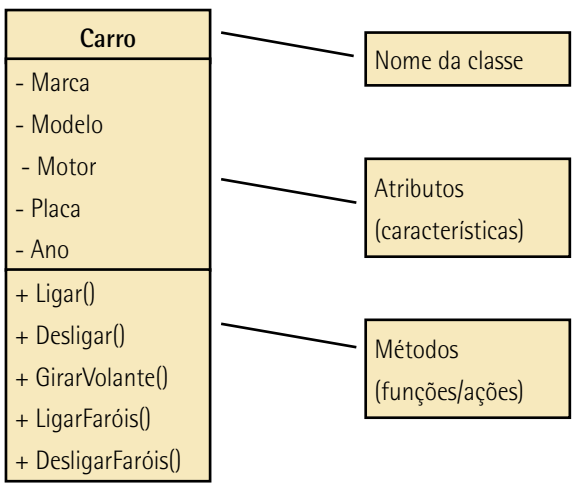

Figura 1 – Representação abstrata do Carro

O que acabamos de fazer na figura anterior foi um trabalho de análise de uma entidade do mundo real e imaginá-la, levando em conta suas características e como seria sua representação ou especificação. Para esse trabalho, damos o nome de abstração.

> Uma classe define os atributos e comportamentos comuns compartilhados por um tipo de objeto. Os objetos de certo tipo ou classificação compartilham os mesmos comportamentos e atributos. As classes atuam de forma muito parecida com um cortador de molde ou biscoito, no sentido de que você usa uma classe para criar ou instanciar objetos.
>
> [...]
>
> Atributos são as características de uma classe visíveis externamente. A cor dos olhos e a cor dos cabelos são exemplos de atributos (SINTES, 2002, p. 8).

Quando temos mais de um objeto, é importante observar a relação entre eles. Esses objetos podem existir independentemente entre si e se precisarem interagir, o farão por meio de mensagens. São as mensagens que fazem com que outro objeto realize alguma operação. Um segundo e importante modo de interação ocorre quando os objetos estão profundamente ligados entre si, de forma que o comportamento de um objeto depende do comportamento de outro. Um objeto pode conter outro dentro de si ou viver em simbiose entre si.

Podemos continuar nosso trabalho de abstração de dados por um aspecto mais biológico. Veja a seguir um exemplo:

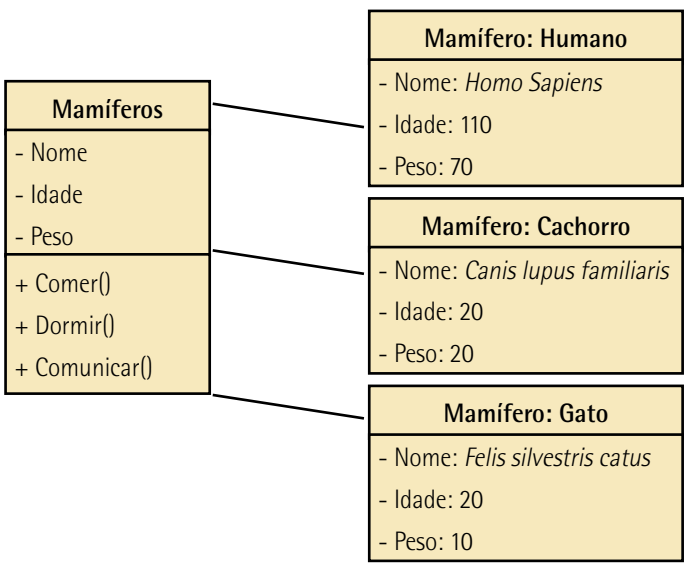

Figura 2 – Exemplo de instanciação de uma classe

Pensando nos animais mamíferos, temos uma classe principal que possui atributos (características) e métodos (comportamentos) próprios da classe mamíferos. Porém, esses métodos e atributos se

materializam pelos animais mais específicos, como cães, gatos e seres humanos. Podemos dizer, portanto, que as classes Humano, Cachorro e Gato são instâncias. Um objeto é uma instância de uma classe (SINTES, 2002, p. 7).

Quando um objeto faz alguma coisa, executa uma ação quando recebe uma mensagem ou responde a uma mudança de estado, ele é chamado de comportamento.

Seguindo essa linha de pensamento, sabemos que esses animais possuem órgãos em seus copos, tal como o coração (veja a figura a seguir).

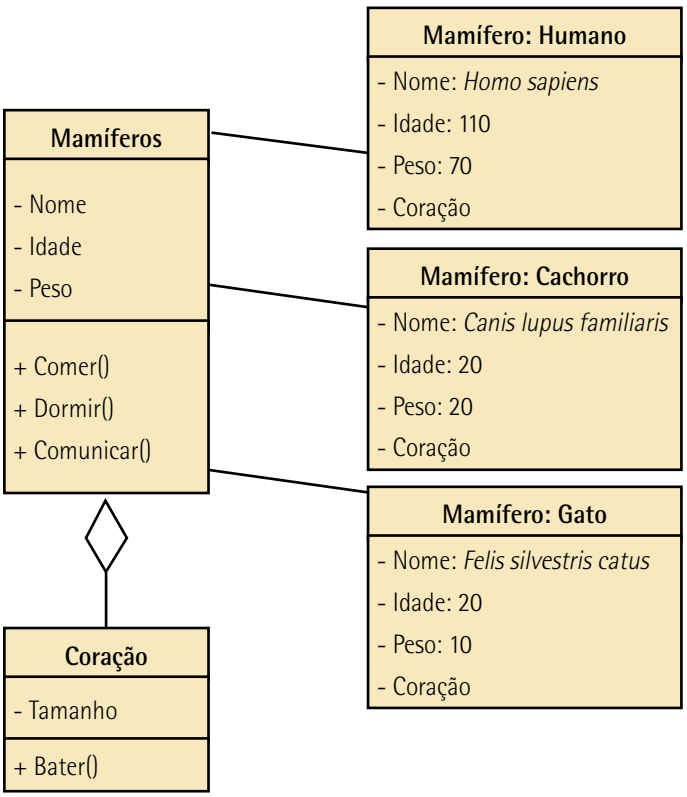

Figura 3 – Exemplo de classe encapsulada

Um órgão como o coração possui características e funções próprias; logo, podemos abstraí-lo e modelá-lo como uma classe específica.

Essa classe Coração irá compor uma característica da classe Mamíferos que, embora faça uso do Coração (assim como as demais classes que herdam suas características), não precisa conhecer seu funcionamento para utilizá-lo. Basta saber que o Coração precisa ter um tamanho específico (para que caiba no animal apropriado) e que ele deve bater, por exemplo.

Dessa forma, damos o nome de **encapsulamento** à técnica de utilizar classes prontas como característica de outra classe de forma a não ter a necessidade, nem a dependência, dos detalhes de seu funcionamento para que tal classe possa ser utilizada.

Encapsulamento é a característica da OO de ocultar partes independentes da implementação. O encapsulamento permite que você construa partes ocultas da implementação do *software*, que atingem uma funcionalidade e ocultam os detalhes de implementação do mundo exterior (SINTES, 2002, p. 22).

A herança permite que você baseie a definição de uma nova classe em uma previamente existente. Assim, podemos definir uma **superclasse** chamada Vertebrados na qual estão reunidos os atributos gerais que todas as suas subclasses possuem. No exemplo, as subclasses Mamíferos e Aves possuem um nome e certa quantidade de vértebras, portanto esses atributos ficam na superclasse. Ao mesmo tempo, cada uma das subclasses possui características próprias que as fazem ser diferentes umas das outras. A subclasse Mamífero não precisaria de um atributo tipo de bico, enquanto para as aves esse atributo é uma das características da espécie. Veja a figura:

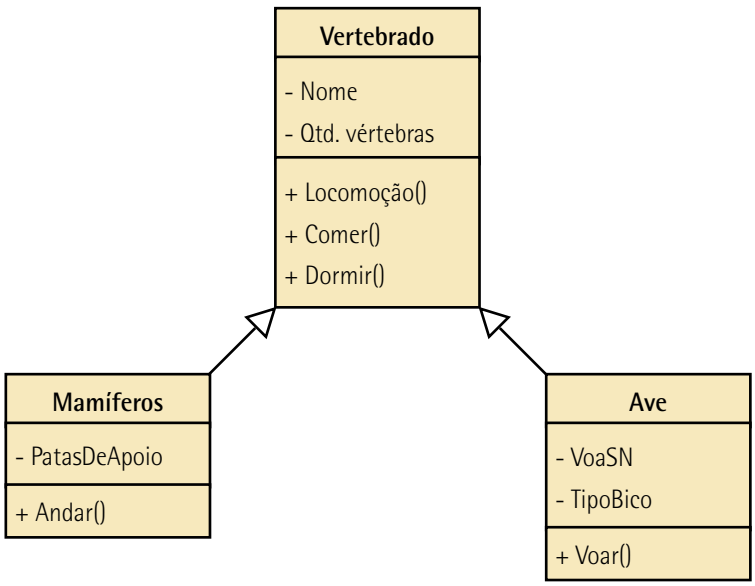

Figura 4 – Exemplo de herança, a superclasse dos vertebrados e duas das suas subclasses

> A herança permite que você baseie a definição de uma nova classe em uma classe previamente existente. Quando você baseia uma classe em outra, a definição da nova classe herda automaticamente todos os atributos, comportamentos e implementações presentes na classe previamente existente (SINTES, 2002, p. 72).

Outro exemplo – um carro funciona da mesma forma: sabemos que dentro dele existe um motor, mas para que o carro funcione não precisamos saber o funcionamento detalhado do motor. Precisamos saber o que é necessário para que ele funcione, ou seja, se o carro tem combustível e tem bateria, giramos a chave e o motor liga! Caso um dia seja necessário trocar o motor do carro, o funcionamento se mantém da mesma forma, mesmo sendo outro motor. Assim, podemos compor um carro com classes independentes que fazem o todo funcionar (veja a figura a seguir). O motor é uma classe independente e pode ser utilizado em outras classes, como carros, barcos e geradores, e isso acontece com a caixa de direção, que pode ser utilizada em outros veículos terrestres, como caminhões, ônibus e tratores.

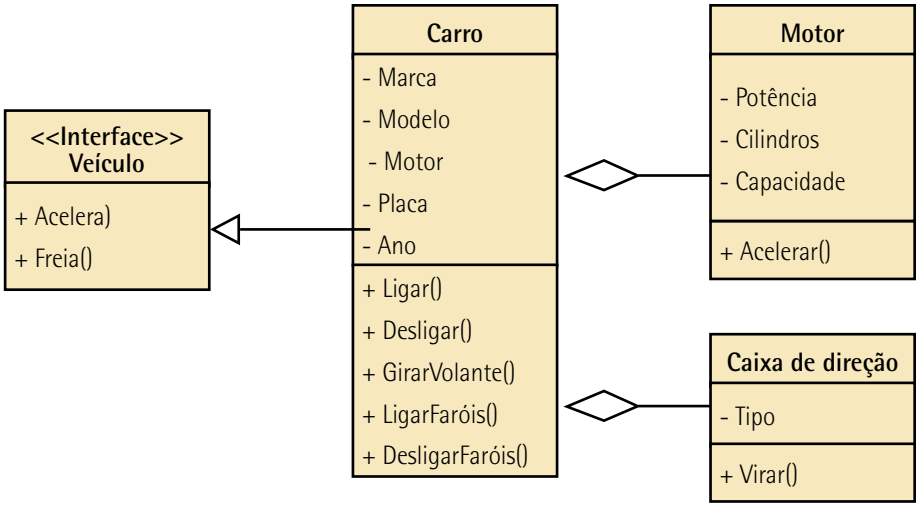

Figura 5 – O uso de uma classe em outra

Outro conceito relevante é o **polimorfismo**. Tal conceito nos diz que coisas diferentes se comportam de maneiras diferentes com a mesma ação.

Se pensarmos nos carros, todos eles funcionam de forma similar, mas existem pequenas alterações de funcionamento (que o motorista não precisa saber para poder usá-los) em determinas ações. A alimentação dos motores com diferentes tipos de combustível exemplifica bem o polimorfismo. O motor é praticamente o mesmo em sua composição, independentemente da categoria, mas responde de maneira semelhante. Veja:

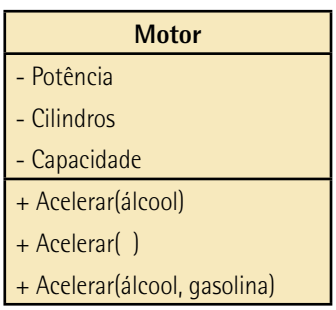

Figura 6 – O polimorfismo permite a utilização de diferentes tipos de combustível para fazer o motor funcionar

Lembrete

Os modelos apresentados neste livro-texto são de um diagrama de classes, que é apenas um dentre os vários diagramas que formam o UML 2.0. Ele faz parte dos diagramas estruturais e somente mostra a estrutura de relação entre as classes.

> Polimorfismo significa muitas formas. Em termos de programação, o polimorfismo permite que um único nome de classe ou nome do método represente um código diferente, selecionado por algum mecanismo automático. Assim, um nome pode assumir muitas formas e como pode representar código diferente, o mesmo nome pode representar muitos comportamentos diferentes (SINTES, 2002, p. 122).

Observação

Os conceitos de encapsulamento, herança, polimorfismo e controle de objetos serão detalhados mais adiante.

Por fim, é importante salientar um conceito simples, mas que é a principal dúvida que surge nas pessoas quando se iniciam os estudos em POO: onde exatamente entram os **objetos**? Classes e objetos são praticamente a mesma coisa, exceto por um detalhe: a implementação. Assim como nos carros, antes de implementá-los, ou seja, construí-los e montá-los, é necessário projetá-los. Fase de projeto é onde se definem as características e ações de cada peça do carro. As classes são definições de como o programa irá funcionar.

Uma vez concluído o projeto, passa-se para a fase de construção, na qual teremos no final o projeto montado e pronto para uso! Assim é o objeto em POO: uma classe que será instanciada (colocada em memória) para o uso.

Tais conceitos ficarão mais claros à medida que os estudos em POO avançarem, pois cada um tem seu momento de uso.

Lembrete

Estudar programação de *software* é uma mistura de arte com exatas! Embora se necessite conhecer muito bem os conceitos e teorias, somente a prática levará o estudante ao verdadeiro aprendizado!

Resumo

Nesta unidade, conhecemos um pouco da história e alguns conceitos da POO, de uma forma bem abrangente. Vimos que a POO não é algo novo, mas que por ser uma técnica amplamente utilizada atualmente, leva a pensar que é recente. Para poder planejar, entender e visualizar melhor seus projetos em POO, recomenda-se aprender também UML (*Unified Modeling Language*), que auxilia a visualizar um projeto OO de diversas maneiras. Para o nosso curso, o importante é entender o diagrama de classes, pois em POO o principal conceito é a classe.

As classes são compostas de atributos e métodos; os atributos descrevem as características da classe, e os métodos, as ações que a classe executa. Uma classe é um molde dos objetos; ela pode gerar vários objetos que funcionam conforme determinado pela classe durante a execução do programa.

A POO, por meio das técnicas de herança e polimorfismo, pode economizar código e padronizar o *software*, reaproveitando o esforço feito no desenvolvimento dos códigos escritos anteriormente ao projeto.

Exercícios

Questão 1. (Enade 2005) A orientação a objetos é uma forma abstrata de pensar em um problema utilizando-se conceitos do mundo real e não, apenas, conceitos computacionais. Nessa perspectiva, a adoção do programa orientado a objetos implica necessariamente que:

A) Os usuários utilizem as aplicações de forma mais simples.

B) Os sistemas sejam encapsulados por outros sistemas.

C) Os programadores de aplicações sejam mais especializados.

D) Os objetos sejam implementados de maneira eficiente e simples.

E) A computação seja acionada por trocas de mensagens entre objetos.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: não é possível determinar o uso das aplicações pelos usuários somente pela forma como foi construída uma solução, seja ela orientada a objetos ou não.

B) Alternativa incorreta.

Justificativa: apesar de um dos conceitos importantes de orientação a objetos ser o de encapsulamento, não há obrigatoriedade de encapsular sistemas dentro de outro sistema.

C) Alternativa incorreta.

Justificativa: não temos como determinar se há maior ou menor necessidade de especialização dos profissionais em função da escolha do paradigma de desenvolvimento escolhido (OO, estruturado, eventos).

D) Alternativa incorreta.

Justificativa: apesar do desejo de que toda solução informatizada desenvolvida seja bem implementada, não temos como afirmar que o fato de adotarmos a orientação a objetos resultará em implementações de classes e objetos de forma simples e eficiente, porque isto não depende do paradigma escolhido, e sim da maneira mais simples e eficiente de planejar a implementação da solução (abstração).

E) Alternativa correta.

Justificativa: ao adotarmos o paradigma de orientação a objetos temos de, obrigatoriamente, elaborar a solução, baseados no fato que a OO funciona somente por meio de trocas de mensagens entre objetos de classes.

Questão 2. (DPE/SP 2010) A cidade de São Paulo, que possuía uma população de 10.000.000 de habitantes, teve um aumento de mais de 2.000.000 de novos habitantes.

Na associação da frase acima aos conceitos de modelagem orientada a objetos, é correto afirmar que **São Paulo, população** e **aumento**, referem-se, respectivamente, a:

A) classe, objeto, instância de classe.

B) objeto, atributo, implementação por um método do objeto.

C) classe, objeto, atributo.

D) objeto, instância, atributo.

E) classe, objeto, associação pelo método de agregação.

Resolução desta questão na plataforma.