Universidade Federal de Juiz de Fora
Centro de Gestão do Conhecimento Organizacional
Diretoria de Sistemas de Informação

Introdução à Programação Orientada a Objeto

2008

Sumário

Introdução

Para usar qualquer linguagem, precisamos saber construir algoritmos. Para usar os modernos ambientes e linguagens de desenvolvimento, precisamos saber também sobre o modelo de Orientação a Objeto (OO), já que quase todas as linguagens hoje têm, em maior ou menor grau, alguma relação com este modelo de programação - inclusive as linguagens mais específicas para Internet (Java ou PHP, por exemplo). Por isso é tão importante entendermos bem este assunto.

Embora se aceite que a primeira linguagem com características de OO foi a Simula 67, este conceito começou a se tornar mais popular entre os profissionais de informática na década de 70, através das pesquisas no PARC (*Palo Alto Research Center*), laboratório da Xerox em Palo Alto, Califórnia, que resultaram na criação da linguagem SmallTalk. Isto é fácil de ser testado: uma simples procura na Internet pelo termo "Simula 67" gera algumas dezenas de milhares de resultados, enquanto que, procurando "SmallTalk" obtém-se milhões de *links*... Mas mesmo assim, esta linguagem não obteve muito sucesso comercial, ficando apenas com a "honra" de ter apresentado o modelo de OO aos desenvolvedores. Outras linguagens, como C++, C#, Object Pascal e Java têm sido muito mais usadas, possivelmente por não serem tão "puras" como SmallTalk e terem adequado a OO ao que havia de mais consagrado na programação mais "tradicional".

A idéia da OO é tratar elementos no processamento de dados da mesma forma que no mundo real. Em nosso cotidiano, consideramos as coisas materiais como objetos (carros, por exemplo), que têm seus atributos (características descritivas como marca, modelo, cor, peso, etc.) e que sabem - têm o método para - realizar determinadas ações (no caso dos carros: fazer curvas, movimentar rodas, etc.). Queria-se usar essa abordagem nas linguagens para computador, visualizando quaisquer elementos que pudessem ser modelados dentro de um programa como se fossem objetos. Nasceu assim a Programação Orientada a Objetos (*Object Oriented Programming* - OOP). Durante algum tempo, o modelo foi pouco prático, pelo purismo citado, por falta de ferramentas mais produtivas e até pela reação normal a qualquer novidade. De meados da década de 1990 para cá, porém, surgiram vários ambientes de desenvolvimento no mercado que alcançaram êxito por terem unido a OOP às facilidades dos sistemas operacionais gráficos, o que garante aprendizado acelerado e criação de programas de forma mais simples, rápida e atrativa. Mais recentemente, com o modelo de OO atingindo popularidade no desenvolvimento específico para *web*, algumas linguagens (PHP, por exemplo) foram incorporando os conceitos de classes, objetos, herança, etc.

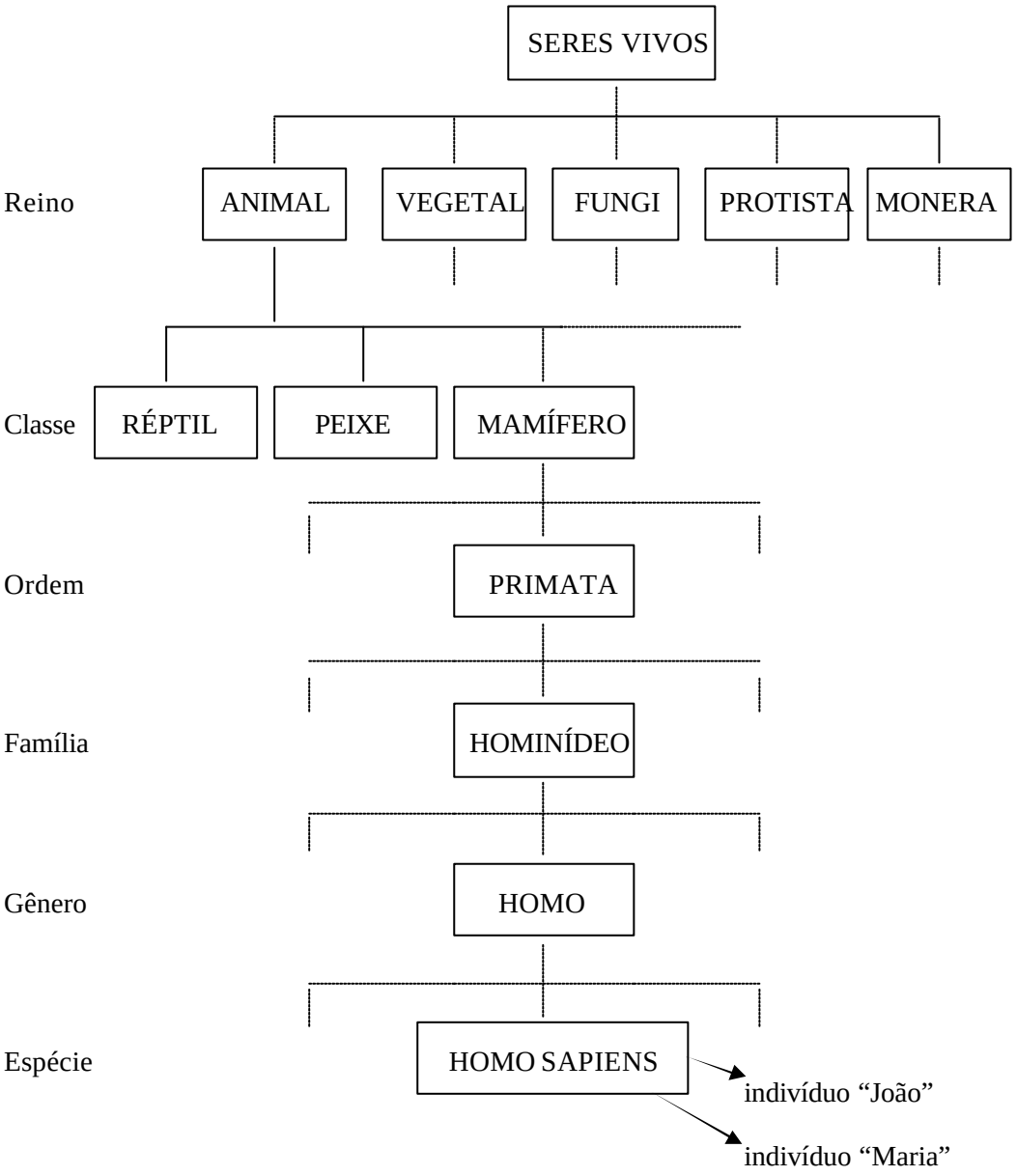
Para compreender como funciona a idéia de OO, vamos recorrer a um velho modelo biológico. Parece estranho usar isso para explicar algo tão aparentemente tecnológico. Mas será muito mais fácil assimilar os conceitos de classes e objetos, se fizermos um paralelo com a classificação biológica sugerida por Lineu, em meados do século XVIII.

A hierarquia taxonômica de Lineu

A taxonomia de Lineu é um sistema formal de classificação dos seres vivos baseado em uma estrutura hierárquica simples, do mais geral ao mais específico. A hierarquia básica continha as seguintes divisões:

Reino
  Classe
    Ordem
      Família
        Gênero
          Espécie

Depois desta idéia inicial, outras categorias (Filo, entre Reino e Classe, por exemplo) foram incluídas na lista acima, de acordo com os avanços científicos; mas, para simplificar, vamos nos ater apenas à hierarquia original. A classificação de Lineu poderia ser representada graficamente conforme a figura abaixo. Obviamente, este esquema não está completo, as linhas pontilhadas significam outras ramificações não mostradas. Para efeitos didáticos, seguiremos apenas o "caminho" até o Homo Sapiens:



Cada retângulo na divisão acima tem suas próprias características e as características dos retângulos ligados acima dele, até chegar ao primeiro (SERES VIVOS). Vamos dar uma olhada na Classe MAMÍFERO. Ela contém as características comuns a todos os mamíferos (o fato de se alimentar mamando no início da vida, por exemplo), mas também tem todas as características de ANIMAIS, que, por sua vez, contém todas as características de SERES VIVOS. Dizemos (em jargão OO) que MAMÍFERO herda todas as características de ANIMAL, e também as características comuns a todos os seres vivos, presentes no primeiro retângulo. Todas as divisões "descendentes" (abaixo e ligadas a MAMÍFERO) herdarão, também, as características dele. Isto é o conceito básico de HERANÇA, que faz com que uma determinada

característica seja implementada apenas uma vez e reutilizada daí por diante (o que se conhece como "reutilização de código" e é um dos maiores benefícios da OO).

Repare que a "árvore" acima começa em SERES VIVOS (o conceito mais geral, poderíamos dizer abstrato) e segue até o HOMO SAPIENS (uma espécie bem definida, específica, concreta), de onde podemos extrair indivíduos (em jargão OO: instanciar objetos). Um homem (João) ou uma mulher (Maria) são indivíduos da espécie HOMO SAPIENS, tal como o FUSCA de placa XYZ-1234 poderia ser um OBJETO de uma CLASSE de carros de passeio.

Um indivíduo faz coisas (falar, por exemplo), porque sabe como fazê-lo. Ou seja: tem o MÉTODO de como falar. Além disso, possui características que o descrevem, tais como seu nome ou estatura (em OO chamamos estas características de ATRIBUTOS).

Classes de Objetos

Transportando os conceitos vistos acima para o mundo OO, cada retângulo na figura passa a ser tratado como uma CLASSE (não confunda com a Classe de Lineu de Peixes, Mamíferos, etc.) e cada indivíduo que tenha sido criado a partir de uma classe é conhecido como um OBJETO dessa classe. Um objeto é uma INSTÂNCIA de sua classe.

Portanto, as classes podem ser encaradas como um modelo para objetos que possuem um mesmo comportamento. Uma classe é definida com vários atributos (por exemplo: nome, altura, largura, etc.) e "conhece" métodos (para criar, destruir, abrir, fechar, etc.) que podem ser acionados. Note que as definições de atributos e métodos estão nas classes, mas, normalmente, serão usadas nos seus objetos (salvo algumas características específicas da classe, das quais falaremos depois). Quando começamos a modelar objetos, uma classe pode ser muito genérica - abstrata - (veículos, por exemplo). Mas, a partir destas, podemos criar classes cada vez mais concretas (carros de passeio, por exemplo).

Encapsulamento

ENCAPSULAMENTO é a capacidade de conter ("escondido" do mundo exterior) tudo que as classes necessitam para serem usadas. Por exemplo, um carro pode fazer as rodas girarem, sem que nós precisemos necessariamente saber como ele o faz. Nós apenas aceleramos, o que seria equivalente a uma ação para chamar o método de girar as rodas. Assim, temos acesso ao objeto, sem tomar conhecimento do funcionamento interno necessário para que ele nos obedeça. Nós dizemos o que queremos, e não como deve ser feito. Futuras melhorias no desenvolvimento do que está encapsulado na classe normalmente não altera o uso dos objetos. Por analogia, imagine que a nova versão do carro do exemplo tenha passado por uma alteração que aumentou a economia de combustível (equivale a dizer que os objetos desta classe ficaram melhores, mas sua forma de uso não mudou).

Essa capacidade é outra das grandes vantagens da OO. Conseguir colocar dentro do objeto (conforme especificado na classe) tudo que ele precisa para existir e se comportar, sem que sejam necessários procedimentos adicionais externos, o que auxilia a compreensão e o uso das estruturas programadas. Assim, os objetos de uma classe sabem tudo o que é preciso para que existam e tenham o comportamento que deles se espera.

Herança

Como já falamos, classes mais específicas são geradas a partir de classes mais genéricas. Herança é a habilidade que uma classe tem em manter (herdar) o comportamento (atributos e métodos) da classe da qual foi derivada. Criamos assim uma SUBCLASSE, onde podemos adicionar novas características. No

caso de termos uma classe de veículos genérica, podemos criar uma subclasse de caminhonetes, à qual adicionamos o atributo tração (2 ou 4 rodas). A classe original (veículos) passa a ser conhecida como SUPERCLASSE em relação à subclasse. Se alguma característica mudar na superclasse, mudará também nas suas subclasses (exceto nos casos onde tenhamos usado o polimorfismo - explicado a seguir - para reescrever parte da subclasse).

Polimorfismo

POLIMORFISMO é a característica que permite que duas ou mais classes (portanto, seus objetos instanciados) respondam de forma diferente a um mesmo pedido. No nosso caso real de exemplo (carros), poderíamos ter um mesmo método (fazer o veículo rodar, por exemplo) ligeiramente diferente para a classe de carros de passeio e para a classe de caminhonetes (por poderem ter tração nas 4 rodas). Ou seja, podemos usar um mesmo nome de método com comportamento diferente nas duas classes, o que evita termos herdado algo inadequado e não podermos fazer nada a respeito. Se não tivéssemos polimorfismo, a classe de caminhonetes do exemplo teria que manter o método "fazer o veículo rodar" antigo para a tração de 2 rodas e ter acrescentado o método "fazer o veículo rodar B" para a tração de 4 rodas... Além disso, é possível manter dois ou mais métodos com o mesmo nome dentro de uma mesma classe, desde que recebendo parâmetros diferentes (isso é um tipo especial de polimorfismo chamado de SOBRECARGA).

Direção por Eventos

Passando do mundo real ao mundo dos computadores, ao invés de carros, vamos usar como exemplo os botões de um programa. Todos os botões nos sistemas operacionais gráficos têm alguns atributos: um pequeno texto, um tipo de fonte, um tamanho, uma localização. É um ótimo candidato a objeto em qualquer linguagem que use o conceito. Mas não é só com o fato de ser um objeto que temos que nos preocupar; este objeto vai responder a atitudes do usuário. Vai responder a EVENTOS.

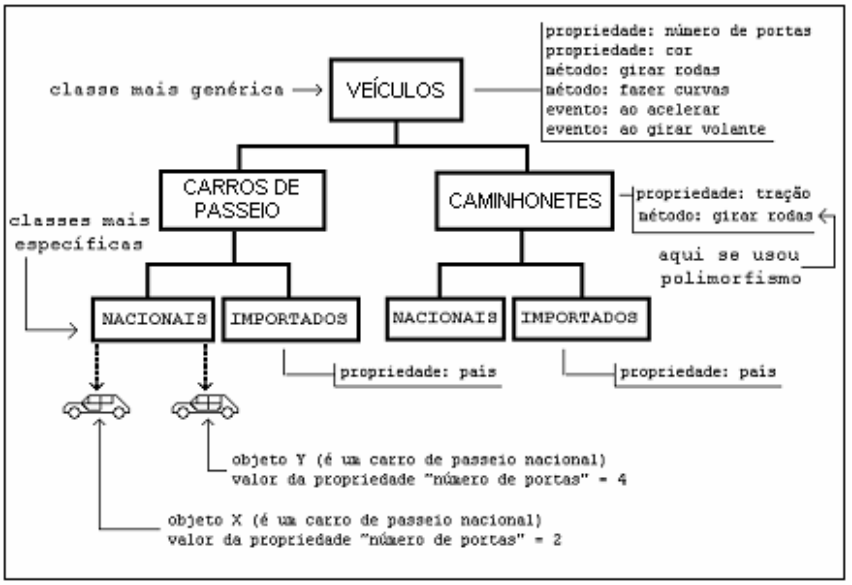
Embora os eventos em si não sejam elementos teóricos específicos de OO, na prática da programação os usamos a todo instante para detectar ações do usuário, disparando assim algum código a partir de um clique do mouse em um botão, ou ao teclar algo, ou ao surgir uma figura na tela, etc. Para isso, os objetos devem, de alguma forma, saber de acontecimentos sobre eles mesmos. Um botão deve saber se foi clicado ou teve o mouse passando por cima dele.

As antigas linguagens procedurais forçavam o usuário a uma seqüência rígida que ele precisava cumprir para realizar uma tarefa. Tome como exemplo um programa para DOS feito em uma linguagem tradicional para preencher uma tela de cadastro. O usuário deveria pular de campo em campo e, quando chegasse ao fim, confirmaria ou não a inclusão. Em uma janela de um sistema operacional gráfico ou em um aplicativo *web* mais moderno, com janelas, botões, menus, barras e o uso livre do mouse para pular para qualquer parte, devemos pressupor uma maior interação do usuário com o programa. Para responder aos atos deste usuário, o programa deve reconhecer estes acontecimentos, ou seja: deve reconhecer os eventos.

Atente para o fato de que a forma de se detectar eventos e responder aos mesmos varia muito de linguagem para linguagem e deve ser estudado de acordo com o ambiente escolhido.

Um Exemplo Visual

Observe a figura a seguir. Vemos uma hierarquia de tipos de veículos para fazer uma analogia com a classificação biológica vista anteriormente. Qualquer coisa pode ser modelada usando-se a Orientação a Objetos. No exemplo anterior os SERES VIVOS eram a divisão mais genérica, assim como tudo é veículo no exemplo abaixo. Todas as demais classes derivam da classe genérica.



Veja que, na modelagem acima, optou-se por fazer com que NACIONAIS e IMPORTADOS fossem subclasses de CARROS DE PASSEIO e CAMINHONETES. Quando este modelo for codificado, não poderemos ter duas classes chamadas NACIONAIS... Deveremos chamar, por exemplo, uma de PasseioNacional e outra de CaminhoneteNacional. O mesmo com os IMPORTADOS. Em geral, usa-se o singular para o nome da classe. Ao invés disso, poderíamos ter optado por uma modelagem onde o país fosse apenas um atributo da classe VEICULO e não houvesse classe NACIONAL nem IMPORTADO. Qual a melhor modelagem? Essa resposta não é tão simples; vai depender de como o modelo será implementado, como será programado, que tipo de evolução terá, etc. Infelizmente, não é raro definirmos um modelo de classes que julgamos perfeito e, depois de algum tempo de uso, descobrirmos que temos problemas com ele. Obviamente, os modelos podem ser alterados, mas o custo pode ser alto. O ideal é que se estude com muita cautela o problema e se façam as fases de análise/projeto com bastante cuidado para evitar ter de mudar todo um sistema por conta de erros de modelagem.