

Aula 12: Introdução a Orientação a Objetos

Professor(a): Virgínia Fernandes Mota

<http://www.dcc.ufmg.br/~virginiaferm>

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA



Origem da Orientação a Objetos

- A orientação a objetos tem sua origem nos anos 60 na Noruega, com Kristen Nygaard e Ole-Johan Dahl, no Centro Norueguês de Computação.
- Através da linguagem Simula 67, foram introduzidos os conceitos de classe e herança (que veremos em breve).
- A orientação a objetos foi mais bem conceituada no laboratório da Xerox, em Palo Alto, sendo refinada numa sequência de protótipos da linguagem Smalltalk.
- O líder desse projeto foi Alan Curtis Kay, considerado um dos criadores do termo "programação orientada a objetos".

Origem da Orientação a Objetos

- Alan Kay começou a programar em Simula depois de conhecer um inovador programa chamado Sketchpad.
- A partir dos conceitos desse sistema, como também dos seus conhecimentos em Biologia e Matemática, Alan Kay formulou sua analogia "algébrico-biológica".
- Ele lançou o postulado de que o computador ideal deveria funcionar como um organismo vivo.
- Alan Kay também observou que o conceito de objetos tinha enorme potencial como uma ferramenta cognitiva: havia uma boa correspondência com a maneira de pensar das pessoas sobre o mundo.

Origem da Orientação a Objetos

- Alan Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagissem entre si, estabelecendo os seguintes princípios da orientação a objetos:
 - Qualquer coisa é um objeto.
 - Objetos realizam tarefas através da requisição de serviços.
 - Cada objeto pertence a uma determinada classe.
 - Uma classe agrupa objetos similares.
 - Um classe possui comportamentos associados ao objeto.
 - Classes são organizadas em hierarquias.

- O que temos então é um novo paradigma de programação.
- Na computação, paradigmas explicam como os elementos que compõe um programa são organizados e como interagem entre si.

Programação orientada a objetos (POO) x Programação estruturada

- Programação orientada a objetos x Programação estruturada.
- Ambas as formas de programação possuem seus altos e baixos.
- Ambas possuem vantagens e desvantagens.
- É possível integrar os dois paradigmas!

Programação orientada a objetos (POO) x Programação estruturada

- Um sistema orientado a objetos é dividido em componentes e não mais em processos.
- Exemplo: Sistema financeiro.
 - **POO**: Teríamos um objeto de fornecedores, por exemplo, onde todas as funções estariam agrupadas no objeto e em nenhum outro lugar.
 - **Estruturada**: As rotinas e funções de fornecedores estariam espalhadas em todo o sistema, como em contas a pagar, contas a receber, cadastro, etc.

Programação orientada a objetos (POO) x Programação estruturada

- Imagine agora o cadastro de fornecedores, com todas as suas rotinas e funções:
 - **Estruturada:** Se você futuramente precisar alterar algum dado, função ou propriedade, o que em seu programa será afetado? O que terá que ser reestruturado? Imagine você voltando à fase de testes e analisando todo o seu sistema até ter certeza que a alteração que você fez não desencadeou listas de alterações que você terá que fazer em todo o sistema...
 - **POO:** as propriedades, funções e rotinas do objeto fornecedores estão todas em um único objeto, encapsulados, facilitando essa necessidade futura de alterações e atualizações.

Programação orientada a objetos (POO) x Programação estruturada

- Reutilização de código:
 - **Estruturada:** É possível a reutilização de código na programação estruturada porém, em muitos casos você será obrigado a utilizar o famoso "copiar e colar".
 - **POO:** Com a POO você é capaz de elaborar um relacionamento entre diversos componentes, estabelecendo comunicação entre eles e facilitando assim, e muito a reutilização de código, além da facilidade de se poder herdar atributos e comportamentos de outros objetos.

Programação orientada a objetos (POO) x Programação estruturada

- Mas qual é a melhor? Depende, vamos discutir mais um pouco.
- **Atenção: Até aqui discutimos os paradigmas e não as linguagens de programação!!!**

Programação orientada a objetos (POO) x Programação estruturada



Programação orientada a objetos (POO) x Programação estruturada

- A linguagem C é a principal representante da programação estruturada: baixo nível.
- A sua principal utilização, devido ao baixo nível, é em programação para sistemas embarcados ou outros em que o conhecimento do hardware se faz necessário para um bom programa. E também quando a velocidade é essencial.
- **IMPORTANTE:** a programação estruturada, quando bem feita, possui um desempenho superior ao que vemos na programação orientada a objetos.

Programação orientada a objetos (POO) x Programação estruturada

- Isso ocorre pelo fato de ser um paradigma sequencial, em que cada linha de código é executada após a outra, sem muitos desvios, como vemos na POO.
- Além disso, o paradigma estruturado costuma permitir mais liberdades com o hardware, o que acaba auxiliando na questão do desempenho.

Programação orientada a objetos (POO) x Programação estruturada

- Entretanto, a programação orientada a objetos traz outros pontos que acabam sendo mais interessantes no contexto de aplicações modernas.
- Como o desempenho não é uma das grandes preocupações na maioria das aplicações (devido ao poder de processamento dos computadores atuais), a programação orientada a objetos se tornou muito difundida.
- Essa difusão se dá muito pela questão da reutilização de código e pela capacidade de representação do sistema muito mais perto do que veríamos no mundo real.

Os 4 pilares da Programação Orientada a Objetos

- Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma.
- Para isso, a linguagem precisa atender a quatro tópicos bastante importantes:
 - Abstração
 - Encapsulamento
 - Herança
 - Polimorfismo

- A **Abstração** consiste em um dos pontos mais importantes dentro de qualquer linguagem Orientada a Objetos.
- São três pontos que devem ser levados em consideração nessa Abstração.
 - Identidade
 - Propriedade/Atributo
 - Métodos

- O primeiro ponto é darmos uma **identidade** ao objeto que iremos criar.
- A identidade deve ser única.
- Nessas linguagens, a identidade do objeto não pode ser repetida dentro do pacote, e não necessariamente no sistema inteiro.

- A segunda parte diz respeito a características do objeto.
- No mundo real qualquer objeto possui elementos/características que o definem: suas **propriedades** ou seus **atributos**.
- Por exemplo, as propriedades de um objeto *Cachorro* poderiam ser *Tamanho*, *Raça* e *Idade*.

- A terceira parte é definirmos as ações que o objeto irá executar.
- Essas ações, ou eventos, são chamados **métodos**.

- O **encapsulamento** é uma das principais técnicas que define a programação orientada a objetos.
- "Segurança": Caixa preta.
- A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados *getters* e *setters*, que irão retornar e definir o valor da propriedade, respectivamente.
- Exemplo: Quando apertamos o botão ligar da televisão, não sabemos o que está acontecendo internamente → métodos que ligam a televisão estão encapsulados.

- O reuso de código é uma das grandes vantagens da programação orientada a objetos.
- Muito disso se dá por uma questão que é conhecida como **Herança**.
- Para entendermos essa característica, vamos imaginar uma família:
 - a criança está herdando características de seus pais. Os pais, por sua vez, herdaram algo dos avós, o que faz com que a criança também o faça, e assim sucessivamente.

- Na orientação a objetos, a questão é exatamente assim:
 - O objeto abaixo na hierarquia irá herdar características de todos os objetos acima dele, seus "ancestrais".
 - A herança a partir das características do objeto mais acima é considerada herança direta, enquanto as demais são consideradas heranças indiretas.
 - Por exemplo, na família, a criança herda diretamente dos pais e indiretamente dos avós e dos bisavós.
- A Herança varia bastante de linguagem para linguagem.

- Outro ponto essencial na programação orientada a objetos é o chamado **polimorfismo**.
- O polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai.
- Exemplo: temos um objeto genérico "Eletrodoméstico". Esse objeto possui um método, ou ação, "Ligar()". Temos dois objetos, "Televisão" e "Geladeira", que não irão ser ligados da mesma forma. Assim, precisamos, para cada uma das classes filhas, reescrever o método "Ligar()".

Os 4 pilares da programação Orientada a Objetos

- Esses quatro pilares são essenciais no entendimento de qualquer linguagem orientada a objetos e da orientação a objetos como um todo.
- Cada linguagem irá implementar esses pilares de uma forma, mas essencialmente a mesma coisa.
- Apenas a questão da herança que pode trazer variações mais bruscas, como a presença de *herança múltipla*.
- Além disso, o encapsulamento também é feito de maneiras distintas nas diversas linguagens, embora os *getters* e *setters* sejam praticamente onipresentes.

Exemplos de Linguagens Orientadas a Objetos

- Há uma grande quantidade de linguagens de programação orientada a objetos no mercado atualmente.
- Mais utilizadas no momento: Java, C#, C++, Python.
- Cada uma delas possui uma abordagem diferente do problema que as torna muito boas para alguns tipos de aplicações e não tão boas para outros.

Exemplos de Linguagens Orientadas a Objetos - Java

- O Java é, muito provavelmente, a linguagem de programação mais utilizada no mercado atual.
- O Java implementa os quatro pilares de forma bastante intuitiva, o que facilita o entendimento por parte do desenvolvedor.
 - Abstração: é implementado através de classes, que contêm propriedades e métodos, de forma bastante simples.
 - Encapsulamento: é realizado através de propriedades privadas, auxiliadas por métodos especiais getters e setters.
 - Herança: simples e com possibilidades de usar Interfaces.
 - Polimorfismo: atributo @Override.

Exemplos de Linguagens Orientadas a Objetos - C#

- O C# é uma linguagem de uso geral e especialmente criada para utilização com a orientação a objetos.
- Vale ressaltar que, em C#, tudo é um objeto (herda da classe *object*).
 - Abstração: segue a ideia do Java.
 - Encapsulamento: um pouco diferente devido a implementação dos métodos *getter* e *setter*. A nomenclatura também é um pouco diferente. A variável que realmente guarda o valor do dado é chamada atributo, enquanto a propriedade é o elemento que realmente acessa aquele dado do mundo externo.
 - Herança: simples e com possibilidades de usar Interfaces.
 - Polimorfismo: Baseado em métodos virtuais. Palavra-chave *override*.

Exemplos de Linguagens Orientadas a Objetos - C++

- O C++, por sua vez, é uma linguagem mais baixo nível, e permite muito mais liberdades com o hardware.
- Como ele foi derivado imediatamente do C, o C++ permite a utilização de ponteiros, por exemplo, que irão trabalhar diretamente com a memória.
- Além disso, o C++ pode utilizar diretamente todas as bibliotecas C.
 - Abstração: implementa classes.
 - Encapsulamento: sentido de privado e público. Métodos *getter* e *setter*.
 - Herança: Permite Herança múltipla.
 - Polimorfismo: Próximo ao C#.

- Python: linguagem de script orientada a objetos que é muito utilizada em pesquisas científicas devido a sua velocidade.
- Object Pascal (também conhecida como Delphi, devido ao nome de sua IDE), que já caiu em desuso, apesar do grande número de sistemas mais antigos que a utilizam.
- Objective-C: linguagem de preferência para desenvolvimento de aplicações para os sistemas da Apple, como iPhone e iPad. (Foi atualizada para Swift).
- Ruby: voltada para o desenvolvimento web.
- Visual Basic .NET, muito utilizada até pouco tempo, mas também caindo em desuso, principalmente devido ao avanço do C# em popularidade.

- **Classe:** Uma classe é o agrupamento de objetos com a mesma estrutura de dados (definida pelos atributos ou propriedades) e comportamento (operações), ou seja, classe são as descrições dos objetos!

- **Objeto:** De maneira geral, Objeto é uma classe sendo estanciada. De maneira mais Conceitual, um objeto é algo distinguível que contém atributos (ou propriedades) e possui um comportamento. Cada objeto tem uma identidade e é distinguível de outro mesmo que seus atributos sejam idênticos.

- **Atributo:** O conjunto de propriedades da classe. Alguns autores preferem distinguir o mesmo como Variável.

- **Métodos:** O conjunto de funcionalidades da classe. Para cada método, especifica-se sua assinatura, composta por:
 - *Nome:* um identificador para o método.
 - *Tipo:* quando o método tem um valor de retorno, o tipo desse valor.
 - *Lista de argumentos:* quando o método recebe parâmetros para sua execução, o tipo e um identificador para cada parâmetro.
 - *Visibilidade:* como para atributos, define o quão visível é um método a partir de objetos de outras classes.

- 1 Crie um modelo para um restaurante caseiro. Imagine que o Restaurante Caseiro Hipotético facilite aos seus clientes a divisão dos valores da conta pelo número de clientes. Que dados adicionais deveriam ser representados pelo modelo? Quais operações deveriam ser criadas?
- 2 Escreva um modelo para representar uma lâmpada que está à venda em um supermercado. Que dados devem ser representados por este modelo?
- 3 Imagine uma lâmpada que possa ter três estados: apagada, acesa e meia-luz. Usando o modelo "Lâmpada" como base, escreva o modelo "LampadaTresEstados".
- 4 Inclua, no modelo "Lâmpada", uma operação "estaLigada" que retorne verdadeiro se a lâmpada estiver ligada e falso, caso contrário.
- 5 Crie um modelo Livro que represente os dados básicos de um livro, sem se preocupar com a sua finalidade.
- 6 Usando o resultado do exercício anterior como base, crie um modelo "LivroDeLivraria" que represente os dados básicos de um livro que está à venda em uma livraria.
- 7 Usando o resultado do modelo "Livro" como base, crie um modelo "LivroDeBiblioteca" que represente os dados básicos de um livro de uma biblioteca, que pode ser emprestado a leitores.

Na próxima aula...

O que é Java?