Projetos de Sistemas de Informação

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico

Programação Orientada a Objetos

Responsável pelo Conteúdo:
Prof. Ms. Fábio Peppe Beraldo

Revisão Textual:
Prof. Ms. Claudio Brites

UNIDADE Programação Orientada a Objetos

- Introdução
- OOP
- Características e Conceitos
- Linguagens OOP
- Linguagens dinâmicas OOP

Objetivo de APRENDIZADO

O objetivo do estudo da disciplina Projeto de Sistemas de Informação é habilitá-lo(a) a projetar melhorias de qualidade em produtos e serviços; e, acima de tudo, identificar as vantagens da modelagem de sistemas; uso do modelo de conceituar e construir sistemas; estudo interdisciplinar da utilização desses modelos; modelagem de sistemas, análise e esforços de design; modelagem e simulação de sistemas, tais como a dinâmica do sistema; sistemas de linguagem de modelagem específica.

Primeiramente, é necessária a conscientização de que a exigência – tanto de estudo quanto de avaliação – da disciplina online é, no mínimo, a mesma que a presencial.

Assim, é necessário organizar seu tempo para ler atentamente o material teórico e assistir os vídeos, se houver, inclusive do material complementar.

Organize-se também de forma a não deixar para o último dia a realização das atividades de sistematização (pontuada) e fórum (não pontuado). Podem ocorrer imprevistos e, encerrada a Unidade, encerra-se a possibilidade de obter a nota relativa a cada atividade.

Para ampliar seu conhecimento, bem como aprofundar os assuntos discutidos, pesquise, leia e consulte os livros indicados nas Referências e/ou na Bibliografia.

Caso ocorram dúvidas, contate o professor tutor por meio do Fórum de Dúvidas, local ideal, pois assim a explicação poderá ser compartilhada por todos.

Contextualização

Programação orientada a objetos é uma abordagem para a concepção de sistemas de *software* com módulos reutilizáveis.

Embora as discussões sobre tecnologia orientada a objetos comumente travem nos detalhes de uma língua contra a outra, a verdadeira chave para a abordagem orientada a objetos é que, em primeiro lugar, trata-se de abordagem de modelagem.

Nesta Unidade teremos uma visão de como a análise de requisitos funciona, assim como suas ferramentas.

Introdução

Glossário

Resumidamente, a Programação Orientada a Objetos – *Object Oriented Programming* (OOP) – é uma forma de programação que representa conceitos como "objetos" possuidores de campos de dados (atributos que descrevem o objeto) e procedimentos associados, conhecidos como métodos. Tratam-se de objetos, geralmente instâncias de classes, que são utilizados para interagir uns com os outros, a fim de projetar aplicativos e programas de computador. *C++, Objective-C, Smalltalk, Java, C#, Perl, Python, Ruby e PHP* são exemplos de linguagens de programação orientadas a objetos.

Nesse sentido, programação orientada a objetos é uma abordagem para a concepção de sistemas de *software* com módulos reutilizáveis.

Embora as discussões sobre tecnologia orientada a objetos comumente travem nos detalhes de uma língua contra a outra, a verdadeira chave para a abordagem orientada a objetos é que, em primeiro lugar, trata-se de abordagem de modelagem. Ainda que alardeada pelos defensores zelosos como uma forma revolucionária no desenvolvimento de *softwares*, na realidade a abordagem orientada a objetos é uma extensão lógica das boas práticas de design que retomam o início da programação de computadores. Em outras palavras, orientação a objetos é simplesmente a extensão lógica de técnicas mais antigas, como programação estruturada e tipos de dados abstratos. Por sua vez, um objeto é um tipo de dado abstrato com a adição de polimorfismo e herança.

Em vez de programas com estruturas de códigos e de dados, um sistema orientado a objetos integra ambos usando o conceito de um objeto. Um objeto possui estado (dados) e comportamentos (códigos), que correspondem a coisas encontradas no mundo real. Assim, por exemplo, um programa de gráficos terá objetos como círculo, quadrado, menu etc. Um sistema de compras *on-line* terá objetos como carrinho de compras, clientes, produtos. O sistema de compras exemplificado apoiará comportamentos como a ordem do lugar, fazer o pagamento e oferecer desconto.

Os objetos são concebidos como hierarquias de classe. Assim, por exemplo, com o sistema de compras pode haver classes de alto nível, como produtos eletrônicos, produtos de cozinha e livros. Pode haver refinamentos, tais como produtos eletrônicos (CD *player*, DVD *player* etc.). Essas classes e subclasses correspondem aos conjuntos e subconjuntos em lógica matemática.

OOP

Os objetivos da programação orientada a objetos são:

- Maior compreensão;
- Facilidade de manutenção;
- Facilidade de evolução.

A compreensão global do sistema é aumentada devido ao gap semântico ser menor – a distância entre a língua dos desenvolvedores e a dos usuários. Ao invés de se referir às tabelas de banco de dados e rotinas de programação, as palestras de desenvolvedores abordam termos e referências que o usuário está familiarizado, como objetos de seu domínio de aplicação.

A orientação a objetos facilita a manutenção pelo uso do encapsulamento e ocultação de informações. Uma das fontes mais comuns de erros nos programas é quando uma parte do sistema interfere acidentalmente em outra parte. Por exemplo, nos primeiros dias de programação, era comum que os desenvolvedores usassem declarações *go to* com o objetivo de saltar para locais arbitrários dentro de algumas rotinas e funções. Os críticos chamam isso de código espaguete, porque é desorganizado. Estruturar endereços de programação incentiva a utilização de procedimentos e rotinas. Pode-se saber, por exemplo, que a função raiz quadrada foi separada da função de mísseis de lançamento, e uma mudança para um não poderia afetar o outro.

O próximo passo da orientação a objetos é fundir os tipos de dados abstratos com programação estruturada e dividir os sistemas modulares em objetos que possuem seus próprios dados e são responsáveis por seu próprio comportamento. Esse recurso é conhecido como encapsulamento. Com o encapsulamento, não só as funções raiz quadrada e mísseis de lançamento não interferem umas com as outras, mas também os dados para ambas são divididos de modo que as mudanças em um objeto não podem afetar o outro.

Informação

Além de facilitar a manutenção, o encapsulamento esconde informação adequadamente. Definir *software* como componentes modulares que suportam herança torna mais fácil tanto o reuso de componentes existentes, quanto estender os componentes conforme necessário através da definição de novas subclasses com comportamentos especializados.

Esta capacidade é conhecida como princípio *open closed*. Um módulo é aberto (*open*) se suportar extensão – pode, por exemplo, facilmente modificar o comportamento, adicionar novas propriedades, fornecer valores padrão etc. Um módulo é fechado (*closed*) se possui uma interface estável e bem definida, onde todos os outros módulos com os quais o primeiro têm seus erros reduzidos.

A abordagem orientada a objetos encoraja o programador a colocar os dados de forma não diretamente acessível ao resto do sistema. Em vez disso, os dados são acessados por funções de chamadas especialmente escritos, conhecidos como métodos, que são empacotados com os dados. Esses agem como intermediários para recuperar ou modificar os dados que controlam. A construção dessa programação, que combina dados com um conjunto de métodos para acessar e gerencia-los é chamada de um objeto. A prática de usar sub-rotinas para examinar ou modificar certos tipos de dados também foi utilizada na programação modular não OOP anteriormente ao uso generalizado de programação orientada a objetos.

Um programa orientado a objetos geralmente contém diferentes tipos de objetos, cada um correspondendo a um conceito do mundo real, como uma conta de banco, um jogador de hóquei, ou um trator. Um programa pode conter inúmeras cópias de cada tipo de objeto, uma

para cada um dos objetos do mundo real dos negócios com programa. Por exemplo, pode haver um objeto conta bancária para cada conta do mundo real em um banco particular. Cada cópia do objeto conta bancária seria estruturado tanto nos métodos que oferece para a manipulação ou a leitura de seus dados, embora os dados dentro de cada objeto seriam diferentes, refletindo a distinta história de cada conta.

Os objetos podem ser pensados como encapsuladores, ou seja, os dados dentro de um conjunto de funções destinadas a assegurar que sejam utilizados de forma adequada. Métodos do objeto normalmente incluem travas e salvaguardas específicas para os tipos de dados que o objeto contém. Um objeto também pode oferecer métodos simples de usar e padronizar a realização de operações específicas em seus dados, enquanto esconde os detalhes sobre como essas tarefas são realizadas. Desta forma, as alterações podem ser aplicadas à estrutura interna ou métodos de um objeto sem exigir que o resto do programa seja modificado. Tal abordagem também pode ser utilizada para oferecer métodos padronizados em diferentes tipos de objetos. A título de exemplo, inúmeros e diferentes tipos de objetos podem oferecer métodos de impressão. Cada tipo pode implementar esse método de impressão de uma forma distinta, refletindo os distinguidos tipos de dados que cada um contém, mas todos os diferentes métodos de impressão podem ser chamados da mesma maneira padronizada de outras partes do programa. Essas características tornam-se especialmente úteis quando mais de um programador contribui com o desenvolvimento de um código para um projeto ou quando o objetivo é a reutilização de um mesmo código entre diferentes projetos.

Características e conceitos

Inúmeros autores criaram algumas características que fundamentaram a capacidade da OOP para o trabalho com a maioria das linguagens de programação orientadas a objetos, as quais:

Dynamic dispatch

Quando um método é chamado em um objeto, o próprio objeto determina o código que será executado, observando-se o método em tempo de execução em uma tabela associada ao objeto. Esta característica distingue um objeto de um tipo abstrato de dados (ou módulo), que tem uma implementação fixa (estática) das operações para todas as instâncias. É um método de programação que permite o desenvolvimento de componentes modulares, mostrando-se plenamente eficaz.

Encapsulation (encapsulamento ou multimétodos)

Utilizado para se referir a um dos dois conceitos relacionados, mas distintos e, por vezes, com a combinação desses. Sobre seus significados:

- Um mecanismo de idioma para restringir o acesso a alguns dos componentes do objeto;
- A construção de uma linguagem que facilita o agrupamento de dados com os métodos (ou outras funções) que operam nesses dados.

Alguns pesquisadores de linguagem de programação e acadêmicos usam o primeiro significado sozinho ou em combinação com o segundo, como uma característica distintiva da programação orientada a objetos, enquanto que outras linguagens de programação que fornecem fechamentos lexicais para visualizar o encapsulamento como uma característica da linguagem ortogonal à orientação a objetos.

A segunda definição é motivada pelo fato de que em muitas linguagens OOP os componentes não se agrupam ou são substituídos automaticamente; assim, ocultação de informações é definida como uma noção separada por aqueles que preferem a segunda definição.

Subtype polymorphism (Polimorfismo de subtipo)

É uma forma de polimorfismo de tipo em que um subtipo é um tipo de dados que está relacionado com outro tipo de dados (o supertipo) por alguma noção de "substituibilidade", o que significa que os elementos do programa, geralmente sub-rotinas ou funções, escritos para operar em elementos do supertipo também pode operar em elementos do subtipo.

Object inheritance (Herança de objeto)

Herança é quando um objeto ou classe é baseado em um outro objeto ou classe, usando a mesma aplicação; trata-se de um mecanismo para a reutilização de códigos. As relações de objetos ou classes através de herança da origem a uma hierarquia. Herança foi inventado em 1967 por Simula e não deve ser confundida com subtipos, dado que herança apenas reutiliza implementação e estabelece uma relação sintática, sem necessariamente criar uma relação semântica, ou seja, herança não garante o subtipo comportamental. Para distinguir estes conceitos, subtipagem também é conhecida como herança da interface, enquanto herança é relacionada a aplicação de herança.

Open recursion

Trata-se de uma variável especial, geralmente chamada *this* ou *self* (sintaticamente podendo ser uma palavra-chave), que permite a um método invocar outro método do mesmo objeto. Importante frisar que essa variável possibilita que o método definido de determinada classe possa invocar outro método em alguma subclasse, que é definido posteriormente.

São apresentados a seguir conceitos adicionais, empregados em programação orientada a objetos.

Classes of objects (Classes de objetos)

Uma classe é um modelo extensível para a criação de objetos, fornecendo valores iniciais (variáveis-membro) e implementações de comportamento (funções de membro, métodos). Na maioria das linguagens, "o nome da classe é usado como o nome para a classe" (o próprio modelo), o nome para o construtor padrão da classe (sub-rotina que cria objetos), e como os tipos de objetos gerados pelo tipo, tais conceitos distintos são facilmente confundidos.

Em algumas linguagens as classes são apenas recursos de compilação, dado que novas classes não podem ser declaradas em tempo de execução; enquanto que em outras linguagens as classes geralmente são os próprios objetos (comumente do tipo class ou similar). Nessas linguagens uma classe que cria classes é chamada de metaclasse.

Instance (instância)

Uma instância é uma realização específica de qualquer objeto. Formalmente, instância é sinônimo de objeto, já que cada um determina certo valor (realização), e esses podem ser chamados de um objeto instância; logo, instância enfatiza a identidade distinta do objeto. A criação de uma instância realizada é entendida por instanciação.

Method (Método ou Função-membro)

É uma sub-rotina (procedimento ou função) associada a um objeto e que tem acesso aos seus dados, suas variáveis-membro. Métodos definem o comportamento a ser exibido por instâncias da classe associada em tempo de execução do programa. Métodos têm a propriedade especial de, em tempo de execução, possuir acesso aos dados armazenados em uma instância da classe (ou instância de classe, ou classe de objeto, ou objeto) que estão associados e são capazes de controlar o estado da instância.

Message passing (Passagem de mensagens)

É um conceito de Ciência da Computação que é amplamente utilizado na concepção e implementação de aplicações de softwares modernos; é a chave para alguns modelos de concorrência e programação orientada a objetos.

Passagem de mensagens é uma forma de invocar o comportamento através de algum serviço de intermediação ou infraestrutura. Ao invés de chamar diretamente um processo, sub-rotina ou função pelo nome, como na programação convencional, transmissão de mensagens envia uma mensagem para um processo (que pode ser um ator ou objeto) e conta com o processo e as infraestruturas de apoio para selecionar e invocar o real código a ser executado.

Abstraction (Abstração)

É o processo de separar as ideias de instâncias específicas das ideias no trabalho. Estruturas computacionais são definidas por seus significados (semântica), enquanto escondem os detalhes sobre como funcionam.

Nesse sentido, abstração tenta fatorar detalhes de um padrão comum para que os programadores possam trabalhar em aproximação ao nível do pensamento humano, deixando de fora detalhes que importam na prática, mas são necessários para o problema ser resolvido. Por exemplo, um sistema pode ter várias camadas de captação de diferentes significados e os valores de pormenores são expostos ao programador; camadas de abstração de baixo nível expõem detalhes do *hardware* do computador onde o programa é executado, enquanto que as camadas de alto nível lidam com a lógica de negócios do programa.

Decoupling (Dissociação)

A dissociação refere-se a controles cuidadosos que separam módulos de códigos a partir de casos de usos específicos, o que aumenta a possibilidade de reutilização do código. É usada geralmente para dissociar o encapsulamento, por exemplo, utilizando uma interface de método que um objeto encapsulado deve preencher, em vez de usar a classe do objeto.

Linguagens OPP

Você Sabia ?

Simula (de 1967) é geralmente aceita como a primeira linguagem com as características principais de uma linguagem orientada a objetos. Foi criada para tornar os programas de simulação, onde o que veio a ser chamado de objeto era a representação mais importante de informações.

Smalltalk (desenvolvido entre 1972 e 1980) sem dúvida é o exemplo canônico e aquele com o qual grande parte da teoria de programação orientada a objetos foi desenvolvida. Em relação ao grau de orientação a objetos, as seguintes distinções podem ser mencionadas:

- Linguagens OO "puros": tudo aqui é tratado de forma consistente como um objeto, a partir de primitivas, como personagens e pontuação; todo o caminho até classes inteiras, protótipos, blocos, módulos etc. Foram projetadas especificamente para facilitar, ou mesmo impor, os métodos OO. Exemplos: *Eiffel, Esmeralda, Jade, Obix, Ruby, Scala, Smalltalk, Self;*

- Linguagens projetadas para programação OO, mas com alguns elementos processuais. Exemplos: *Delphi/Object Pascal, C++, Java, C#, VB.NET, Python;*

- Linguagens procedurais: foram estendidas com algumas características OO. Exemplos: *Pascal, Visual Basic (derivada do BASIC), MATLAB, Fortran, Perl, COBOL 2002, PHP, ABAP, Ada 95;*

- Linguagens com a maioria das características dos objetos (classes, métodos, herança, reutilização), mas constituídas de uma forma distintamente originais. Exemplos: *Oberon (Oberon-1 ou Oberon-2);*

- Linguagens com suporte aos tipos de dados abstratos, onde não apresentam todos os recursos de orientação a objetos (às vezes chamadas de linguagens baseadas em objetos). Exemplos: *Modula-2, flexível, CLU.*

Linguagens dinâmicas OPP

Nos últimos anos a programação orientada a objetos tornou-se especialmente popular em linguagens dinâmicas de programação. *Python, Ruby e Groovy* são linguagens dinâmicas construídas sobre princípios OOP, enquanto *Perl e PHP* foram adicionando recursos orientados a objetos desde *Perl 5 e PHP 4*, além de *ColdFusion*, agregando tais recursos desde a versão 6.

O *document object model* de *HTML, XHTML e* documentos *XML* na *Internet* têm ligações para a popular linguagem *JavaScript/ECMAScript*. *JavaScript* talvez seja a linguagem de programação mais conhecida com base em protótipos, que emprega a clonagem a partir de protótipos, em vez de herda-las de uma classe (contraste com a programação baseada em classes). Outra linguagem de script que leva essa abordagem é *Lua*. Antes *ActionScript 2.0* (uma super parcial do *ECMA-262 R3*, também conhecida como *ECMAScript*). Foi suportado apenas um modelo de objeto com base em protótipo.

Material Complementar

PRESSMAN, R. S.**ENGENHARIA De Software**. 6ª ed. Porto Alegre: Grupo A, 2010. (e-book)

STEPHEN R. S. **Engenharia de Software**. 8ª ed. Porto Alegre: Grupo A, 2008. (e-book)

PADUA, W. **Engenharia de Software**, 3ª ed. Rio de Janeiro: Grupo GEN, 2008. (e-book)

KALINOVSKY, **A. Java Secreto: técnicas de descompilação, patching e engenharia reversa**. São Paulo: Pearson, 2009. (e-book)

PFLEEGER, S. L. **Engenharia de Software: teoria e prática** - 2º ed. São Paulo: Pearson, 2009. (e-book)

Referências

KALINOVSKY, **A. Java secreto: técnicas de descompilação, patching e engenharia reversa**. São Paulo: Pearson, 2009.

PADUA, W. **Engenharia de software**. 3. ed. Rio de Janeiro: Grupo GEN, 2008.

PFLEEGER, S. L. **Engenharia de software: teoria e prática**. 2. ed. São Paulo: Pearson, 2009.

PRESSMAN, R. S. **Engenharia de software**. 6. ed. Porto Alegre, RS: Grupo A, 2010.

SCHACH, S. R. **Engenharia de software: os paradigmas clássicos e orientado a objetos**. 7. ed. São Paulo: Bookman, 2009.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Addison-Wesley, 2007.

STEPHEN R. S. **Engenharia de software**. 8. ed. Porto Alegre, RS: Grupo A, 2008.

WAZLAWICK, R. S. **Análise e projeto de sistemas de informação orientados a objetos**. 2. ed. Rio de Janeiro: Elsevier, 2011.

Anotações


Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000

Universidade Cruzeiro do Sul

UNICID Universidade Cidade de S. Paulo

UNIFRAN Universidade de Franca

UDF Centro Universitário

Módulo Centro Universitário