Orientação a objetos

**Introdução**

Evolução do paradigma de programação

Até aqui.....

Programas implementados na base de tarefas executadas uma após a outra, linearmente.

A partir de um teste lógico, determinado trecho de código é executado, ou não.

Também, a partir de um teste lógico, determinado trecho de código é repetido por um número finito de vezes.

Essas são características da ‘Programação Estruturada’ ou ‘Paradigma Estruturado’

Ou ainda ‘Paradigma Imperativo’

Algumas linguagens com esse paradigma:

Pascal

C

A programação orientada a objetos é um paradigma de desenvolvimento no qual a estrutura do software é baseada na iteração entre objetos.

Cada um é responsável por uma tarefa e guarda as suas informações.

A interação se dá por troca de mensagens entre esses objetos.

Ao receber uma mensagem o objeto executa uma ação que esteja prevista.

Um objeto é uma entidade que possui atributos (estado) e métodos (comportamento).

Em determinado contexto (domínio), uma pessoa pode ser um objeto.

Atributos: nome, idade, altura, peso, etc.

Métodos: andar, caminhar, falar, etc.

Um carro também pode ser visto como um objeto.

Atributos: modelo, potência, número de portas,etc;

Métodos: acelerar, freiar, ligar, etc.

Pessoa e carro : Uma pessoa interage com o carro passando comandos a ele. O carro responde aos comandos.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. (Wikipedia)

Os conceitos sobre Orientação a Objetos(OO) começaram no meio de 1960. No ambiente acadêmico.

Explodiu em popularidade no início dos anos 90 com a chegada do Java.

C# chegou em 2002...

O conceito predominante de programação era a chamada programação estruturada.

Consistia basicamente em dividir a tarefa de programação em pequenos blocos (procedimentos ou funções).

O programa é uma série de passos a serem executados um após o outro.

Esse tipo de programação funciona muito bem para programas pequenos com poucas centenas de linhas.

Entretanto, quando o programa começa a se tornar maior, aumenta a dificuldade de manter, evoluir, corrigir, etc.

Era difícil converter o modelo de negócio para o modelo do sistema. Dados e código eram muito entranhados.

A programação orientada a objetos combate justamente essas dificuldades:

Consegue converter de forma natural o modelo de negócio para o modelo do sistema;

Facilita a manutenção e evolução do sistema;

Aumenta a capacidade de reuso de trechos de programas escritos por terceiros.

Entretanto, é importante salientar que ambos paradigmas não são excludentes, e que a Programação Orientada a Objetos(POO) não veio para aniquilar a programação estruturada.

Pelo contrário, a POO abraça e expande o conceito de programação estruturada para horizontes ainda mais amplos.

Ao fim a POO apresenta as seguintes vantagens:

Modelo que mais se aproxima do mundo real.

Facilita ao máximo o reuso.

A manutenção e alteraração de programas é executada mais facilmente.

Objetos são a chave para entender a OO;

Se olharmos em nossa volta, encontraremos vários exemplos de objetos reais:

Celular;

Computador;

Mesa;

Projetor;

Etc.

Os objetos reais possuem duas caracterísicas:

Estado (Atributos);

Comportamento.

Por exemplo, um cachorro

Estado: nome, cor, raça, fome…

Comportamento: latindo, abanando o rabo, comendo…

Uma bicicleta

Estado: modelo, marcha atual, freio, rotação…

Comportamento: mudando de marcha, freando…

Um objeto de software é conceitualmente similar aos objetos reais

Objetos armazenam seu estado em atributos

Correspondentes às variáveis em programação estruturada.

Objetos expõem seu comportamento através de métodos

Correspondentes às funções em programação estruturada.

Classe é um conjunto de objetos:

Características semelhantes;

Comportamento comum;

Interação com outros objetos.

Uma classe é a forma para criação de objetos

Objetos são representações concretas (instâncias) de uma classe

Uma classe é o projeto a partir do qual objetos são criados.

A classe define os atributos (propriedades) e métodos que cada objeto terá.

Cada objeto instanciado terá suas próprias propriedades e métodos.

Como uma classe é estruturada em C#:

Declaração da classe

Atributos (campos do objeto)

Métodos

Classe “Telefone”

Atributos: Fabricante, SO, Cor

Construtores

Métodos: Ligar

A representação de classes em diagramas UML contempla três tipos básicos de informação:

nome da classe;

seus atributos;

seus métodos.

O paradigma de orientação a objetos se sustenta por 4 princípios fundamentais:

Abstração,

Encapsulamento,

Herança,

Polimorfismo.

O entendimento dessas 4 propriedades é fundamental para a utilização dos artifícos da orientação a objetos oferece.

Habilidade de se concentrar nos aspectos essenciais do sistema, ou um contexto qualquer, ignorando o que é supérfluo.

Conceito aplicado a criação de software, partindo do princípio que devemos considerar a essência de cada objeto e não pensar em todos os detalhes de implementação;

Semelhante ao que normalmente fazemos na nossa vida em relação aos objetos que nos rodeiam.

É essencial para identificarmos classes.

Seleção de alguns aspectos de domínio do problema a modelar, desconsiderando os irrelevantes para o nível de abstração em questão.

Se você está construindo uma aplicação para gerenciar o envio de cargas:

Os atributos a serem considerados para a carga seria tamanho e peso.

Se a aplicação é sobre controle de estoque, o atributo cor pode ser interessante.

As propriedades a serem consideradas dependerão do contexto e da necessidade.

O objeto esconde seus dados de outros objetos e permite que os dados sejam acessados por intermédio de seus próprios métodos.

Esconder o estado interno e requerer que toda interação seja feita através de métodos é chamado de encapsulamento.

O encapsulamento permite a visualização de uma entidade de software como uma caixa preta.

Você sabe o que a caixa preta faz, pois conhece sua interface externa e, simplesmente, envia mensagens para a caixa preta, não se preocupando com o que acontece dentro dela; você apenas se preocupa com o fato de que isso aconteça.

Os atributos representam o “estado” do objeto.

O estado deve ser acessível ou modificado somente através de métodos, que seriam a interface provida pelo encapsulamento.

Evita-se alterações acidentais nos atributos de um objeto.

Caso haja alguma alteração nos atributos, temos certeza de qual método foi utilizado.

O padrão é que os atributos sejam privados, e sejam oferecidos os métodos acessores (getters) públicos, e caso a classe seja mutável deve-se fornecer os métodos modificadores (setters).

A linguagem C# disponibiliza uma outra maneira para acessar os atributos: as propriedades.

Uma propriedade, basicamente, agrupa os métodos de consulta e alteração dos atributos.

O relacionamento de Herança define um relacionamento do tipo “é um”

“Mountain Bike é uma bicicleta”.

Indica que uma (a subclasse) das duas classes relacionadas é uma forma especializada da outra (a superclasse)

A superclasse é considerada uma generalização da subclasse.

Diferentes tipos de objetos frequentemente possuem semelhanças com outros

Bicicletas Tandem;

Mountain bikes;

Bicicletas de corrida.

Todas estas bicicletas possuem características de bicicletas

Velocidade atual;

Rotação atual;

Marcha atual.

No entanto, também possuem características diferentes

Tandem possuem dois bancos e guidões;

As de corrida possuem guidão angulado;

Moutain bikes possuem correntes maiores, alterando o peso das marchas.

A orientação a objetos permite que as classes herdem o estado e comportamento comuns a outras classes

Neste exemplo, a classe Bicicleta se torna a superclasse de MountainBike, TandemBike e RoadBike.

Estas agora são consideradas subclasses.

Neste exemplo, o mecanismo de herança permite que MountainBike, TandemBike e RoadBike possuam automaticamente os mesmos atributos e métodos da superclasse Bicicleta

E ainda permite que cada uma das subclasses defina seus atributos e métodos adicionais;

Ou seja, nas subclasses nos preocuparemos em definir apena aquilo que as tornam diferentes.

O código da superclasse não estará disponível no código da subclasse

Então é necessário que ele esteja bem documentado.

Quando uma subclasse possui mais que uma superclasse, usamos o termo herança múltipla;

Também, uma subclasse pode se tornar superclasse de outras.

Podemos pensar sobre herança como algo semelhante a funções

Quando identificamos um trecho de código que se repete várias vezes, criamos uma função com aquele conteúdo;

Quando identificamos várias características em comum em um grupo de classes, podemos criar uma superclasse;

Evita-se código duplicado.

Em uma eventual necessidade de alteração nesses trechos comuns, os desenvolvedores de software só precisam fazer as alterações na superclasse, e, consequentemente, as subclasses herdam as alterações.

Sem a herança, as alterações precisariam ser feitas em todos os arquivos de código-fonte que contêm o trecho de código em questão.

Polimorfismo é o princípio pelo qual uma classe possui um método com o mesmo nome e assinatura (número, tipo e ordem de parâmetros) de um método na sua superclasse.

Também ocorre também quando existem dois métodos com mesmo nome, na mesma classe com e assinaturas diferentes.

O método será escolhido de acordo com o número de parâmetros, tipo ou valor de retorno esperado.

Note que esta decisão ocorre em tempo de compilação.

A grande contribuição do Polimorfismo é permitir que classes forneçam diferentes implementações de métodos que são chamados com o mesmo nome;