Rodrigo Gonçalves Rodrigues

Ulbra Torres

Texto acadêmico do Curso de Laboratório de Programação

‘Introdução

A programação orientada a objetos veio como uma alternativa as características da programação estruturada, tendo o objetivo de aproximar o manuseio das estruturas de um programa ao manuseio de coisas do mundo real. Sendo um paradigma de programação que revolucionou a forma de desenvolver softwares. Baseando-se em conceitos como encapsulamento, herança e polimorfismo, ele oferece um jeito estruturado e modular para o desenvolvimento de sistemas complexos, já que os desenvolvedores podem criar componentes reutilizáveis, significando menos tempo resolvendo problemas e mais tempo aprimorando a experiência do usuário com recursos incríveis. As linguagens que suportam programação orientada a objetos geralmente utilizam herança para reutilização de código e extensibilidade na forma de “classes” ou “Protótipos”. Aquelas que utilizam classes tem dois conceitos principais, classes onde apresenta as definições para o formato de dados e procedimentos disponíveis para um dado tipo ou classe de objeto, conhecidos como métodos de classe.

Objeto que representam instâncias de classes objetos às vezes correspondem a coisas encontradas no mundo real. Às vezes, objetos representam entidades mais abstratas, como um objeto que representa um arquivo aberto. Cada objeto é dito ser uma instância de uma classe específica. Procedimentos no POO são conhecidos como métodos, variáveis também são conhecidas como campos, membros, atributos ou propriedades.

Variáveis de classe pertencem à classe como um todo, há apenas uma cópia de cada variável, compartilhada entre todas as instâncias da classe, variáveis de instância ou atributos , dados que pertencem a objetos individuais, cada objeto tem sua própria cópia de cada variáveis de membro. Objetos são acessados de uma forma semelhante às variáveis com estrutura interna complexa , são efetivamente ponteiros, servindo como referências reais para uma única instância desse objeto na memória, dentro de um heap ou pilha. Eles fornecem uma camada de abstração que pode ser sada para separar o código interno do externo. O código externo pode usar um objeto chamando, um método de instância especifico com um conjunto de parâmetros de entrada, ler uma variável de instância ou escrever em uma variável de instância. Os objetos são criados chamando um tipo especial de método na classe conhecida como construtor.

Encapsulamento

É um dos pilares da POO, que permite agrupar dados e os métodos que operam sobre esses dados em uma única entidade, conhecida como classe.trazendo vantagens como proteção de dados por meio de controle de acesso. A ocultação de detalhes internos com a implementação interna de uma classe poder ser alterada sem afetar as partes do código que a utilizam, desde que a interface pública permaneça a mesma.

O encapsulamento funciona com a utilização de modificadores de acesso para restringir o acesso aos atributos e as métodos de um objeto. Garante somente as classes apropriadas possam acessar as informações e ajuda a assegurar que os atributos e os métodos sejam usados de uma forma consistente e previsível. Os modificadores são o private que só podem ser acessados na própria classe em que foram declarados, protected, podem ser acessado pela própria e subclasses, public podem ser acessados por qualquer classe.

Exemplo:

Aqui temos a classe cliente onde as propriedades “nome” e “numeroIdentificacao” são “public” sendo informações mais públicas sendo utilizadas em várias classes, já as propriedades com modificador “protected” são informações mais sigilosas então só são usadas em algumas classes como da própria loja onde não serão exibidas publicamente.

```
public class Cliente
{
    public string Nome { get; set; }
    public string NumeroIdentificacao { get; set; }

    protected string Endereco { get; set; }
    protected string Contato { get; set; }
```

Herança

É um mecanismo em que uma classe pode herdar as propriedades de outra classe. A classe que herda é chamada da classe filha ou subclasse, e a classe que é herdada é chamada de classe pai ou superclasse. A herança permite que as classes filhas reutilizem o código da classe pai e adicione

novos recursos. A classe filha pode substituir os métodos da classe pai e adicionar métodos e variáveis.

Exemplo:

Aqui temos a superclasse (Pai) chamada “Produto”, onde tem suas subclasses onde possui suas subclasses “ProdutoFisico” e” ProdutoDigital” onde recebem as propriedades do pai e adicionam novas para sua categoria especifica.

```
public abstract class Produto
{
    public string Nome { get; set; }
    public int Codigo { get; set; }
    public decimal Preco { get; set; }

    public Produto(string nome, int codigo, decimal preco)
    {
        Nome = nome;
        Codigo = codigo;
        Preco = preco;
    }
```

```
public class ProdutoFisico : Produto
{
    private double preco;
    public double altura;
    public double largura;
    public double profundidade;

    public double Peso { get; set; }
```

```
    public string Categoria { get; set; }


        {
```

```
public class ProdutoDigital : Produto
{
    private string? nomeprodutodigital;
    private double precoprodutodigital;
    private int codigoprodutodigital;

    public double TamanhoArquivo { get; set; }

    public string Formato { get; set; }
```

Polimorfismo

É como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos, que por mais que apresentam a mesma assinatura, tem comportamento diferente para cada uma das classes derivadas. É um mecanismo por meio do qual selecionados as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução. Com ele os mesmo atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes

Exemplo:

Aqui temos um exemplo de polimorfismo presente no “CalcularPrecoFinal” onde pode ter a mesma assinatura nessas três classes, mas tendo seu comportamento diferente em cada uma, para atender suas exigências.

```
public abstract class Produto
{
```

```
    public string Nome { get; set; }
    public int Codigo { get; set; }
    public decimal Preco { get; set; }

    public Produto(string nome, int codigo, decimal preco)
    {
        Nome = nome;
        Codigo = codigo;
        Preco = preco;
    }

    public abstract decimal CalcularPrecoFinal();
}
```

```
 public class ProdutoFisico : Produto
{
    private double preco;
    public double altura;
    public double largura;
    public double profundidade;
    public double Peso { get; set; }
    public string Categoria { get; set;
    public Dimensoes DimensoesProduto { get; set; }

public override decimal CalcularPrecoFinal()
    {
       double taxaDeImposto = (double)Preco * 0.1;
        double precoComImposto = (double)Preco + taxaDeImposto;
        double precoPorKg = 5;
        double custoEnvio = Peso * precoPorKg;
        double precoTotal = precoComImposto + custoEnvio;
        return (decimal)precoTotal;

       public class ProdutoDigital : Produto
{
    private string? nomeprodutodigital;
```

```
    private double precoprodutodigital;
    private int codigoprodutodigital;

    public double TamanhoArquivo { get; set; }

    public string Formato { get; set; }
public override decimal CalcularPrecoFinal()
    {
         double taxaDeDesconto = (double)Preco * 0.2;
        double precoFinal = (double)Preco - taxaDeDesconto;
        return (decimal)precoFinal;
    }
```

Abstração

É a capacidade de ocultar detalhes irrelevantes ou complexos de um problema e focar nos aspectos essenciais. Implementada por meio das classes, que descrevem os atributos e comportamentos de um grupo de objetos. Classes são modelos ou moldes nos quais surgirão os objetos.As classes definem algumas propriedades e métodos que deverão fazer parte do objeto que vai derivar ou então, os objetos que serão instanciados a partir dela.Traz vários benefícios para como evitar erros, pois permite que os desenvolvedores escondam detalhes de implementação que podem causar problemas.

Exemplo:

Aqui temos o exemplo de abstração na classe “ICarriavel” nos métodos “AdicionarProduto”, “RemoverProduto” e “CalcularProduto” onde são declarados na mesma , mas implementados em outra como na classe Produto.

```
public interface ICarriavel
{
    public  void  AdicionarProduto(Produto produto);
    public  bool RemoverProduto(int codigoproduto);
    public  decimal CalcularTotal();
}
```

```
public class Pedido : ICarriavel
{
    public Cliente Cliente { get; set; }
    public DateTime DataPedido { get; set; }
    public string Status { get; set; }
    public List<Produto> Produtos { get; set; }

        public  Pedido(List<Produto>  produtos,  Cliente  cliente,
DateTime datapedido, string status)
    {
        Produtos = produtos;
        Cliente = cliente;
        DataPedido = datapedido;
        Status = status;
    }

    public void AdicionarProduto(Produto produto)
    {
        if (Produtos.Any(p => p.Codigo == produto.Codigo))
        {
                Console.WriteLine("Já existe um produto com esse
código.");
        }
        else
        {
```

```
            Produtos.Add(produto);
            Console.WriteLine("Produto adicionado com sucesso");
        }
    }

    public bool RemoverProduto(int codigoproduto)
    {
            Produto produtoRemover = Produtos.FirstOrDefault(p =>
p.Codigo == codigoproduto);
        if (produtoRemover != null)
        {
            Produtos.Remove(produtoRemover);
            return true;
        }
        return false;
    }

    public decimal CalcularTotal()
    {
        decimal total = 0;
        foreach (var item in Produtos)
        {
            if (item is ProdutoFisico produtoFisico)
            {
                total += produtoFisico.CalcularPrecoFinal();
            }
            else if (item is ProdutoDigital produtoDigital)
            {
                total += produtoDigital.CalcularPrecoFinal();
            }
        }
        return total;
    }
```

Conclusão

Durante toda a semana de trabalho percebi o quão interessante e útil pode ser a programação orientada a objetos, não ha duvidas que ele vai avançar ao longo do tempo facilitando mais a vida de todos que estão aprendendo , quanto aqueles já experientes.

Por mais que às vezes canse, traga frustração e raiva , no final quando você terminar o código, testar , dai você percebe o quão longe percorreu e ver que por mais que tenha sua dificuldade no início , ele pode te ensinar a ser um programador melhor. E, além disso, ter um apoio dos colegas quando precisa, é bom saber que você não está sozinho nessa caminhada.

Referências Bibliográficas:

https://www.dio.me/articles/a-importancia-da-programacao-orientada-a-objetos-para-desenvolvedores-de-todos-os-niveis

https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?utm_term=&utm_campaign=%5BSearch%5D+%5BPerformance%5D+-+Dynamic+Search+Ads+-+Artigos+e+Conteúdos&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=11384329873&hsa_grp=164240702375&hsa_ad=703853654617&hsa_src=g&hsa_tgt=dsa-2276348409543&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=EAIaIQobChMIo7fN0smuiQMV-FRIAB0htwugEAAYAiAAEgLvmvD_BwE

https://www.dio.me/articles/vantagens-da-programacao-orientada-a-objetos-poo

https://pt.wikipedia.org/wiki/Programação_orientada_a_objetos#:~:text=Programação%20orientada%20a%20objetos%20

https://www.devmedia.com.br/conceitos-e-exemplos-heranca-programacao-orientada-a-objetos-parte-1/18579

https://www.dio.me/articles/programacao-orientada-a-objetos-heranca-x-associacao

https://www.devmedia.com.br/conceitos-e-exemplos-polimorfismo-programacao-orientada-a-objetos/18701
https://blog.grancursosonline.com.br/poo-principal-conceito-de-polimorfismo/

https://coodesh.com/blog/candidates/metodologias/conceitos-de-poo-abstracao-e-encapsulamento/