Conceitos de Programação Orientada a Objetos

SisZoológico

Angelo Dalla Corte Euzebio[1]

RESUMO

A Programação Orientada a Objetos (POO) é essencial na construção de sistemas modernos, pois permite uma estrutura flexível e organizada, favorecendo a reutilização de código e a manutenção eficiente. Neste artigo, serão explorados os conceitos fundamentais da POO — encapsulamento, herança, polimorfismo e abstração — e sua aplicação no desenvolvimento de um sistema de gerenciamento de zoológico. Além disso, serão abordados elementos importantes como propriedades, métodos, construtores e sobrecarga, assim como conceitos avançados, incluindo interfaces, classes abstratas e métodos abstratos. Utilizando C# e a plataforma .NET 8, será demonstrada a implementação desses princípios para garantir um sistema robusto. Dessa maneira, será possível entender como a POO contribui para a criação de um ambiente digital estruturado, promovendo a organização e facilitando futuras expansões do sistema.

Palavras-Chave: programação orientada a objetos; encapsulamento; herança; polimorfismo; abstração; C#; .NET 8; métodos; construtores; sobrecarga; interfaces; classes abstratas; métodos abstratos.

ABSTRACT

Object-Oriented Programming (OOP) is essential in the development of modern systems, as it enables a flexible and organized structure, facilitating code reuse and efficient maintenance. This article explores the fundamental concepts of OOP—encapsulation, inheritance, polymorphism, and abstraction—and their application in the development of a zoo management system. Additionally, it discusses important elements such as properties, methods, constructors, and overloading, as well as advanced concepts including interfaces, abstract classes, and abstract methods. Using C# and the .NET 8 platform, the implementation of these principles will be demonstrated to ensure a robust system. Thus, it will be possible to understand how OOP contributes to the creation of a structured digital environment, promoting organization and facilitating future system expansions.

[1] Graduando no Curso Superior de Tecnologia em Análise e Desenvovimento de Sistemas na Universidade Luterana do Brasil - Campus Torres. Email: angelodallacorte@rede.ulbra.br

Keywords: object-oriented programming; encapsulation; inheritance; polymorphism; abstraction; C#; .NET 8; methods; constructors; overloading; interfaces; abstract classes; abstract methods.

1 INTRODUÇÃO

A Programação Orientada a Objetos (POO) é um dos paradigmas mais utilizados no desenvolvimento de sistemas modernos, pois proporciona uma estrutura modular, facilita a reutilização de código e melhora a manutenção de projetos. Fundamentada em quatro pilares — encapsulamento, herança, polimorfismo e abstração —, a POO permite a criação de soluções organizadas, eficientes e escaláveis.

Este artigo tem como propósito demonstrar a aplicação prática dos princípios da POO por meio do desenvolvimento de um sistema de gerenciamento de zoológico, implementado como uma aplicação console em C#. O projeto utiliza a plataforma .NET 8 e explora conceitos essenciais da orientação a objetos, como classes abstratas, interfaces, propriedades, métodos, construtores e sobrecarga, visando uma abordagem clara e funcional.

Durante a implementação do sistema, buscou-se representar a estrutura administrativa de um zoológico, incluindo o registro e a interação entre animais e funcionários. Esse cenário foi escolhido por possibilitar uma exploração detalhada dos princípios da POO, permitindo definir relações hierárquicas entre classes, implementar comportamentos distintos para objetos e estabelecer regras de acesso adequadas para as propriedades.

Ao longo do estudo, serão analisados trechos do código-fonte da aplicação com o objetivo de demonstrar como os recursos da linguagem C# foram empregados para garantir a correta aplicação dos conceitos da POO, contribuindo para um sistema robusto e bem estruturado.

2 PILARES DA POO

A Programação Orientada a Objetos (POO) organiza o desenvolvimento de software em torno de objetos, facilitando a estruturação e manutenção do código. Seus quatro pilares garantem essa eficiência: o encapsulamento protege os dados, permitindo acesso controlado; a herança permite que uma classe aproveite características de outra, reduzindo redundâncias; o polimorfismo possibilita a adaptação de comportamentos conforme a necessidade; e a abstração simplifica conceitos, destacando apenas aspectos essenciais.

2.1 ENCAPSULAMENTO

Encapsulamento refere-se à ocultação dos dados internos de uma classe, expondo apenas o necessário. Na aplicação, as propriedades de classes como `Animal`, `Funcionario`, `Veterinario` e `Zelador` foram definidas com públicos (`get; set;`), conforme o exemplo abaixo:

```
public string Nome { get; set; }
public int Idade { get; set; }
```

2.2 HERANÇA

Herança permite que classes derivadas reutilizem membros de uma classe base. A classe abstrata `Animal` é estendida pelas classes `Mamifero`, `Ave` e `Reptil`, como no exemplo:

```
class Mamifero : Animal
```

2.3 POLIMORFISMO

Polimorfismo é a capacidade de uma mesma interface servir a diferentes tipos. O método abstrato `EmitirSom()` é implementado de maneiras distintas nas subclasses:

```
public override void EmitirSom() => Console.WriteLine($"{Nome} está emitindo som.");
```

2.4 ABSTRAÇÃO

Abstração foca nos aspectos essenciais de um objeto. As classes `Animal` e `Funcionario` são abstratas, exigindo que as subclasses implementem os métodos definidos:

```
public abstract void EmitirSom();
public abstract void Movimentar();
```

3 PROPRIEDADES E MÉTODOS

As propriedades controlam o acesso aos dados. Os métodos definem comportamentos, como por exemplo em `Zelador`:

```
public void AlimentarAnimal(Animal animal) =>
    Console.WriteLine($"Zelador {Nome} alimentou o animal {animal.Nome} com sucesso.");
```

4 CONSTRUTORES

Construtores são utilizados para inicializar objetos com valores. Um exemplo da classe `Veterinario`:

```
public Veterinario(string nome, int idade) :
   base(nome, idade, "Veterinário") {}
```

5 SOBRECARGA DE MÉTODOS

A sobrecarga permite múltiplas definições de um mesmo método com diferentes parâmetros. Na classe `Veterinario`, temos:

```
public void RealizarTratamento(Animal animal)
public void RealizarTratamento(string nomeAnimal)
```

Já na classe Zelador, temos:

```
public void CuidarHabitat(Animal animal)
public void CuidarHabitat(string nomeAnimal)
```

6 INTERFACES

Interfaces definem contratos que devem ser seguidos por classes. `ITratamentoAnimal` e `ICuidador` são exemplos utilizados:

```
interface ITratamentoAnimal
{
   void RealizarTratamento(Animal animal);
}
```

7 CLASSES E MÉTODOS ABSTRATOS

Classes abstratas fornecem uma base para outras classes. Métodos abstratos exigem implementação nas subclasses:

```
abstract class Funcionario
{
   public abstract void Trabalhar();
}
```

8 CONSIDERAÇÕES FINAIS

A aplicação desenvolvida demonstra a utilização prática dos principais conceitos de Programação Orientada a Objetos - POO. Cada estrutura foi implementada para garantir a reutilização de código, segurança e manutenção eficiente, promovendo um sistema flexível para gerenciamento de zoológico.

REFERÊNCIAS

FOGAÇA, Lucas Rodrigues Schwartzhaupt – Aulas ministradas e videos criados pelo Professor da Ulbra, Polo Torres, no Curso LABORATÓRIO DE PROGRAMAÇÃO;

Stack, Ramon full - https://www.youtube.com/@ramonfullstack Maratonas de C#;

Caelum, https://github.com/caelum/apostila-csharp-orientacao-objetos.