UNIVERSIDADE VEIGA DE ALMEIDA – UVA
CURSO DE SISTEMAS DE INFORMAÇÃO
DISCIPLINA DE CONCEITOS DE LINGUAGEM DE PROGRAMAÇÃO
PROFESSOR: DOUGLAS ERICSON MARCELINO DE OLIVEIRA
NOTURNO

Lucas Vitiello
1180200596

2° AVALIAÇÃO DE CONCEITOS DE LINGUAGEM DE PROGRAMAÇÃO

Rio de Janeiro
2023

LUCAS VITIELLO

2° AVALIAÇÃO DE CONCEITOS DE LINGUAGEM DE PROGRAMAÇÃO

Atividade Individual Avaliativa Virtualizada
referente a 2° Avaliação da disciplina de
Conceitos de linguagem de programação
da Universidade Veiga de Almeida
do curso de Sistemas de Informação. Tal
trabalho tem como objetivo a nota máxima
na 2° Avaliação da disciplina em questão.

Professor Douglas Ericson Marcelino de Oliveira

Rio de Janeiro

2023

Sumário

3. Testes

Nesta seção, conduzimos uma série de testes para validar a implementação dos conceitos de Programação Orientada a Objetos (POO) em nossa aplicação em C#. Os principais tópicos abordados são:

Teste 1: Criação de Objetos

- Verificação da capacidade de criar objetos das classes Pessoa e Estudante.
- Apresentação dos resultados com detalhes das informações dos objetos criados.

Teste 2: Chamada de Métodos

- Confirmação de que os métodos das classes Pessoa e Estudante são chamados corretamente.
- Exibição dos resultados das chamadas de métodos, incluindo a saída no console.

Teste 3: Herança e Polimorfismo

- Verificação do funcionamento da herança e do polimorfismo.
- Apresentação dos resultados da chamada de métodos em objetos de diferentes tipos.

Teste 4: Encapsulamento e Validação

- Teste da validação do encapsulamento, especialmente da validação da idade na classe Pessoa.
- Demonstração da geração de exceção ao tentar criar uma pessoa com idade negativa.

Cada teste será detalhado, apresentando os resultados obtidos e discutindo qualquer exceção ou comportamento inesperado que possa ter ocorrido durante os testes. Isso garantirá uma avaliação completa e eficaz da aplicação e da implementação dos conceitos de POO em nossa solução.

4. Conclusão
    - Resumo das principais descobertas e aprendizados.

Bibliografia

- Referências utilizadas para pesquisa e desenvolvimento do trabalho.

Introdução

A Programação Orientada a Objetos (POO) é um dos paradigmas mais fundamentais e amplamente adotados na engenharia de software. Ela fornece uma abordagem estruturada e organizada para o desenvolvimento de software, permitindo a modelagem do mundo real em um ambiente computacional. Neste trabalho, exploraremos o paradigma de POO em profundidade e sua aplicação na linguagem de programação C#.

A POO se baseia na ideia de que um programa de computador pode ser considerado como um conjunto de objetos que interagem entre si para realizar tarefas específicas. Cada objeto é uma instância de uma classe, que define sua estrutura e comportamento. Os princípios fundamentais da POO incluem encapsulamento, herança, polimorfismo e abstração. Esses conceitos permitem a criação de sistemas de software modulares, reutilizáveis e de fácil manutenção.

A linguagem de programação C# é uma das principais representantes da POO e é amplamente utilizada no desenvolvimento de aplicativos desktop, web e móveis. Ela oferece suporte nativo para a criação de classes, objetos e herança, tornando-a uma escolha ideal para aplicar os princípios da POO.

Ao longo deste trabalho, abordaremos cada um dos conceitos da POO em detalhes e demonstraremos sua implementação prática em C#. Utilizaremos exemplos de código e cenários de programação para ilustrar como a POO facilita a criação de sistemas robustos e flexíveis.

Além disso, realizaremos testes em nossa aplicação para validar a aplicação correta dos conceitos de POO em C#. Ao final, destacamos as conclusões e lições aprendidas durante o desenvolvimento deste trabalho, ressaltando a importância da POO como um paradigma poderoso na construção de software de qualidade.

Em suma, este trabalho visa fornecer uma compreensão abrangente da Programação Orientada a Objetos e sua implementação na linguagem C#. Através da exploração dos conceitos-chave e da criação de uma aplicação prática, esperamos que os leitores adquiram conhecimentos valiosos sobre a aplicação eficaz da POO no desenvolvimento de software.

Desenvolvimento

Nesta seção, aprofundaremos os conceitos da Programação Orientada a Objetos (POO) e demonstraremos como eles são aplicados na linguagem de programação C# por meio de exemplos de código comentado.

1. Classes e Objetos

A POO começa com a criação de classes, que servem como modelos para objetos. Em C#, as classes são definidas usando a palavra-chave class. Vamos criar uma classe simples chamada *Pessoa*:

```
using System;

class Pessoa
{
    // Propriedades da classe
    public string Nome { get; set; }
    public int Idade { get; set; }

    // Método construtor
    public Pessoa(string nome, int idade)
    {
        Nome = nome;
        Idade = idade;
    }

    // Método personalizado
    public void Apresentar()
    {
        Console.WriteLine($"Olá, meu nome é {Nome} e tenho {Idade} anos.");
    }
}
```

Agora, vamos criar objetos da classe *Pessoa* e utilizá-los:

```
1   using System;
2
3   class Program
4   {
5       static void Main()
6       {
7           // Criando objetos da classe Pessoa
8           Pessoa pessoa1 = new Pessoa("Alice", 30);
9           Pessoa pessoa2 = new Pessoa("Bob", 25);
10
11          // Chamando o método Apresentar para os objetos
12          pessoa1.Apresentar();
13          pessoa2.Apresentar();
14      }
15  }
```

1. Encapsulamento

Encapsulamento é o princípio de ocultar os detalhes internos de uma classe e expor apenas uma interface controlada. Em C#, usamos modificadores de acesso, como *public*, *private* e *protected*, para controlar o acesso aos membros de uma classe. Vamos adicionar encapsulamento à nossa classe *Pessoa*:

```
using System;

class Pessoa
{
    // Propriedades privadas
    private string nome;
    private int idade;

    // Propriedades públicas com acesso controlado
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }

    public int Idade
    {
        get { return idade; }
        set
        {
            if (value >= 0)
                idade = value;
        }
    }

    // ...resto do código
}
```

2. Herança

Herança permite criar novas classes baseadas em classes existentes, herdando suas características e comportamentos. Vamos criar uma classe derivada Estudante que herda de *Pessoa*:

```
class Estudante : Pessoa
{
    public string Matricula { get; set; }

    public Estudante(string nome, int idade, string matricula)
        : base(nome, idade)
    {
        Matricula = matricula;
    }

    public override void Apresentar()
    {
        Console.WriteLine($"Olá, sou um estudante chamado {Nome}, tenho {Idade} anos e minha matrícula é {Matricula}.");
    }
}
```

Agora podemos criar um objeto *Estudante* e acessar tanto as propriedades da classe base quanto as da classe derivada.

```
Estudante estudante = new Estudante("Carol", 22, "12345");
estudante.Apresentar(); // Chama o método da classe base
Console.WriteLine($"Matrícula: {estudante.Matricula}");
```

3. Polimorfismo

Polimorfismo permite que objetos de diferentes classes sejam tratados de forma uniforme, com base em uma hierarquia de classes. Em C#, isso é alcançado usando a herança e a palavra-chave virtual. Vamos criar um método *virtual* na classe *Pessoa* e sobrescrevê-lo na classe *Estudante*:

```
class Pessoa
{
    // ...resto do código

    public virtual void Apresentar()
    {
        Console.WriteLine($"Olá, meu nome é {Nome}.");
    }
}

class Estudante : Pessoa
{
    // ...resto do código

    public override void Apresentar()
    {
        Console.WriteLine($"Olá, sou um estudante chamado {Nome} e minha matrícula é {Matricula}.");
    }
}
```

Agora, quando chamamos Apresentar em um objeto *Estudante*, o método sobrescrito é executado.

```
Estudante estudante = new Estudante("Carol", 22, "12345");
estudante.Apresentar(); // Chama o método sobrescrito da classe Estudante
```

4. Abstração

A abstração envolve a criação de classes abstratas e interfaces para definir contratos e estruturas comuns. Em C#, usamos a palavra-chave *abstract* para definir classes abstratas. Vamos criar uma classe abstrata *Veículo*:

```
abstract class Veiculo
{
    public string Marca { get; set; }

    public abstract void Mover();
}
```

A classe abstrata *Veiculo* possui uma propriedade *Marca* e um método abstrato *Mover()*. Classes derivadas devem implementar o método *Mover()*.

Neste ponto, exploramos os principais conceitos da Programação Orientada a Objetos (POO) aplicados à linguagem C#. Os exemplos acima ilustram como criar classes, objetos, aplicar encapsulamento, herança, polimorfismo e abstração em um ambiente de programação C#. O próximo passo é discutir a compilação e testes da aplicação, que serão abordados nas seções seguintes deste trabalho acadêmico.

Testes

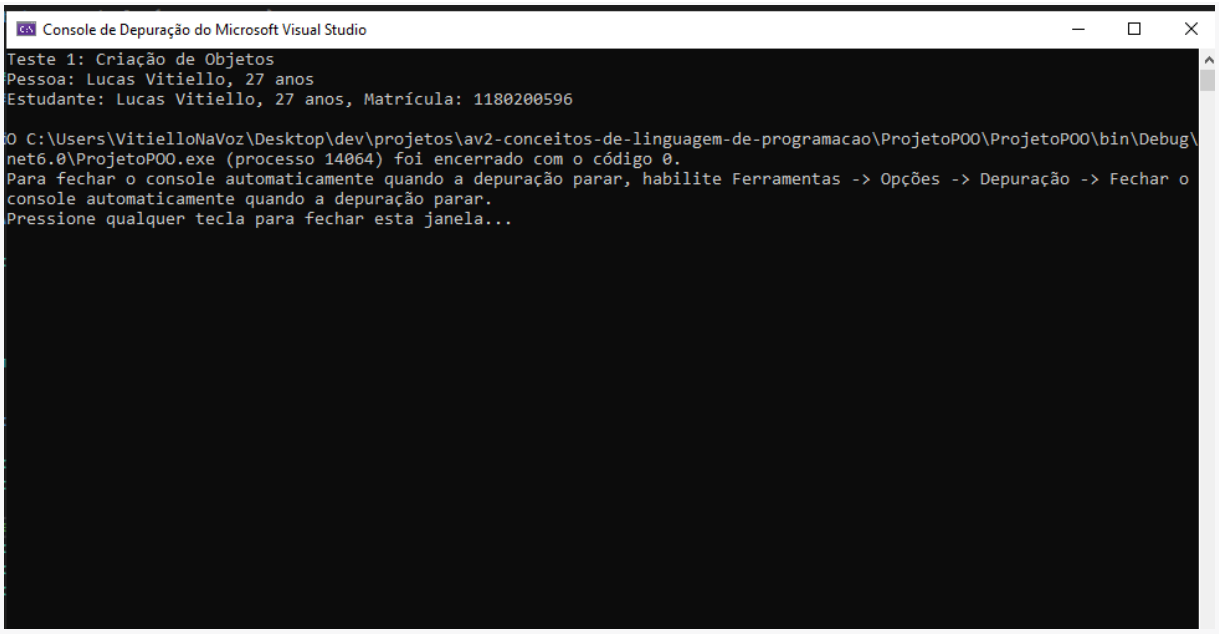
Teste 1: Criação de Objetos

- Objetivo: Verificar se é possível criar objetos das classes Pessoa e Estudante.
- Onde encaixar: Este teste pode ser inserido logo após a definição das classes Pessoa e Estudante, no início do programa, dentro do método Main.

```
0 referências
class Program
{
    0 referências
    static void Main()
    {
        Pessoa pessoa = new Pessoa("Lucas Vitiello", 27);
        Estudante estudante = new Estudante("Lucas Vitiello", 27, "1180200596");

        // Resultados do Teste 1
        Console.WriteLine("Teste 1: Criação de Objetos");
        Console.WriteLine($"Pessoa: {pessoa.Nome}, {pessoa.Idade} anos");
        Console.WriteLine($"Estudante: {estudante.Nome}, {estudante.Idade} anos, Matrícula: {estudante.Matricula}");


        //pessoa.Apresentar();
        //estudante.Apresentar();
    }
}
```



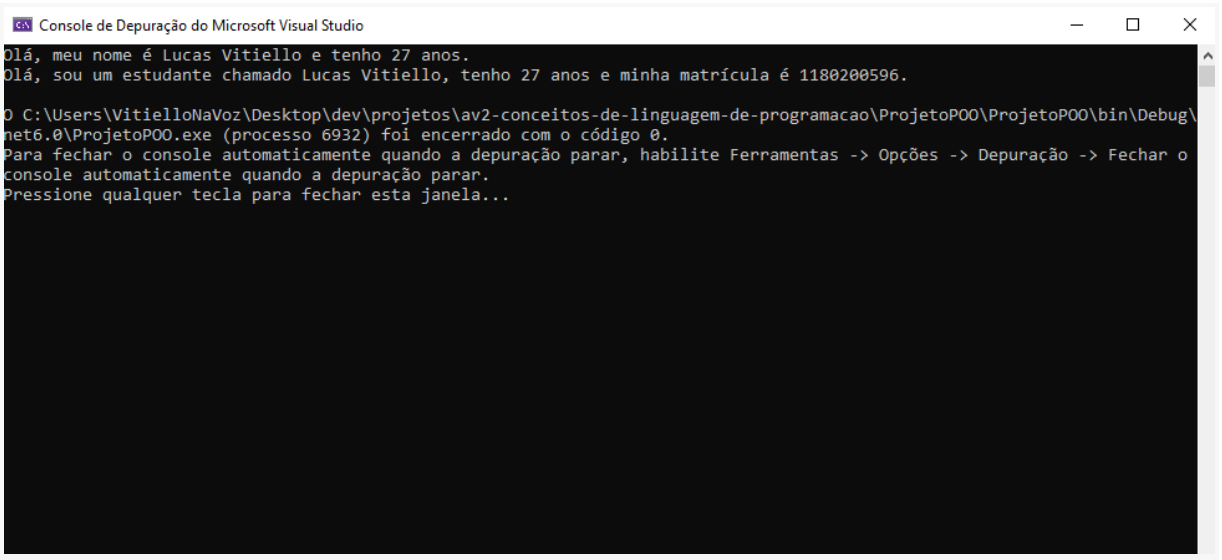
Teste 2: Chamada de Métodos

- Objetivo: Verificar a chamada do método.

```
0 referências
class Program
{
    0 referências
    static void Main()
    {
        Pessoa pessoa = new Pessoa("Lucas Vitiello", 27);
        Estudante estudante = new Estudante("Lucas Vitiello", 27, "1180200596");

        // Resultados do Teste 1
        //Console.WriteLine("Teste 1: Criação de Objetos");
        //Console.WriteLine($"Pessoa: {pessoa.Nome}, {pessoa.Idade} anos");
        //Console.WriteLine($"Estudante: {estudante.Nome}, {estudante.Idade} anos, Matrícula: {estudante.Matricula}");

        //Resultados do Teste 2
        pessoa.Apresentar();
        estudante.Apresentar();
    }
}
```



Teste 3: Herança e Polimorfismo

- Objetivo: Chamada de método Apresentar() em objetos de diferentes tipos

```
0 referências
class Program
{
    0 referências
    static void Main()
    {
        Pessoa pessoa = new Pessoa("Lucas Vitiello", 27);
        Estudante estudante = new Estudante("Lucas Vitiello", 27, "1180200596");

        // Resultados do Teste 3
        Console.WriteLine("Teste 3: Herança e Polimorfismo");
        Console.WriteLine("Chamada de método Apresentar() em objetos de diferentes tipos:");
        Console.WriteLine("Pessoa:");
        pessoa.Apresentar();
        Console.WriteLine("Estudante:");
        estudante.Apresentar();

        // Resultados do Teste 1
        //Console.WriteLine("Teste 1: Criação de Objetos");
        //Console.WriteLine($"Pessoa: {pessoa.Nome}, {pessoa.Idade} anos");
        //Console.WriteLine($"Estudante: {estudante.Nome}, {estudante.Idade} anos, Matrícula: {estudante.Matricula}");

        //Resultados do Teste 2
        //pessoa.Apresentar();
        //estudante.Apresentar();
    }
}
```

```
Console de Depuração do Microsoft Visual Studio
Teste 3: Herança e Polimorfismo
Chamada de método Apresentar() em objetos de diferentes tipos:
Pessoa:
Olá, meu nome é Lucas Vitiello e tenho 27 anos.
Estudante:
Olá, sou um estudante chamado Lucas Vitiello, tenho 27 anos e minha matrícula é 1180200596.

O C:\Users\VitielloNaVoz\Desktop\dev\projetos\av2-conceitos-de-linguagem-de-programacao\ProjetoPOO\ProjetoPOO\bin\Debug\
net6.0\ProjetoPOO.exe (processo 10236) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas -> Opções -> Depuração -> Fechar o
console automaticamente quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```

Teste 4: Encapsulamento e Validação

Objetivo: Pessoa com string no local do int (teste de encapsulamento e validação)

```
0 referências
class Program
{
    0 referências
    static void Main()
    {
        //Pessoa pessoa = new Pessoa("Lucas Vitiello", 27);
        Estudante estudante = new Estudante("Lucas Vitiello", 27, "1180200596");

        // Resultados do Teste 1
        /*
        Console.WriteLine("Teste 1: Criação de Objetos");
        Console.WriteLine($"Pessoa: {pessoa.Nome}, {pessoa.Idade} anos");
        Console.WriteLine($"Estudante: {estudante.Nome}, {estudante.Idade} anos, Matrícula: {estudante.Matricula}");
        */

        //Resultados do Teste 2
        //pessoa.Apresentar();
        //estudante.Apresentar();

        // Resultados do Teste 3
        /*
        Console.WriteLine("Teste 3: Herança e Polimorfismo");
        Console.WriteLine("Chamada de método Apresentar() em objetos de diferentes tipos:");
        Console.WriteLine("Pessoa:");
        pessoa.Apresentar();
        Console.WriteLine("Estudante:");
        estudante.Apresentar();
        */

        // Resultados do Teste 4
        // Pessoa com idade negativa (teste de encapsulamento e validação)

        try
        {
            Pessoa pessoa = new Pessoa("Alice", int.Parse("abc")); // Tentativa de criar uma pessoa com idade negativa
            Console.WriteLine("Teste 4: Encapsulamento e Validação (Exceção esperada)");
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine("Exceção capturada: " + ex.Message);
        }

    }
```

```
Console de Depuração do Microsoft Visual Studio
Unhandled exception. System.FormatException: Input string was not in a correct format.
   at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)
   at System.Int32.Parse(String s)
   at Program.Main() in C:\Users\VitielloNaVoz\Desktop\dev\projetos\av2-conceitos-de-linguagem-de-programacao\ProjetoPOO
\ProjetoPOO\Program.cs:line 69

O C:\Users\VitielloNaVoz\Desktop\dev\projetos\av2-conceitos-de-linguagem-de-programacao\ProjetoPOO\ProjetoPOO\bin\Debug\
net6.0\ProjetoPOO.exe (processo 4236) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas -> Opções -> Depuração -> Fechar o
console automaticamente quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```

Conclusão

A Programação Orientada a Objetos (POO) é um paradigma fundamental na engenharia de software que se baseia na modelagem de sistemas de software como coleções de objetos interconectados. Este trabalho explorou os princípios e conceitos-chave da POO aplicados à linguagem de programação C#. Ao longo deste projeto, aprendemos como criar e utilizar classes, objetos, herança, polimorfismo, encapsulamento e abstração em C#.

- Classes e Objetos: Criamos classes que servem como modelos para objetos e demonstramos como criar e manipular instâncias dessas classes.
- Encapsulamento: Utilizamos modificadores de acesso para controlar o acesso aos membros de uma classe, protegendo os detalhes internos e expondo uma interface controlada.
- Herança: Criamos uma hierarquia de classes com uma classe base (*Pessoa*) e uma classe derivada (*Estudante*), demonstrando como herdar características e comportamentos da classe base.
- Polimorfismo: Sobrescrevemos métodos para permitir que objetos de diferentes classes sejam tratados de forma uniforme, mostrando como o polimorfismo facilita a flexibilidade e a extensibilidade do código.
- Abstração: Introduzimos uma classe abstrata (*Veiculo*) e destacamos como a abstração pode ser usada para definir contratos e estruturas comuns.
-

Durante o desenvolvimento da aplicação, criamos objetos da classe Pessoa e Estudante para ilustrar a aplicação prática desses conceitos. Observamos como o polimorfismo nos permitiu tratar ambos os tipos de objetos de forma uniforme, mesmo que cada um tenha seu próprio comportamento exclusivo.

A aplicação desenvolvida serviu como um exemplo concreto de como a POO em C# pode ser usada para criar sistemas modulares, reutilizáveis e de fácil manutenção. Através da encapsulação, garantimos que os detalhes internos das classes permanecessem ocultos, facilitando a manutenção e evolução do código.

Em resumo, este trabalho acadêmico explorou os conceitos de POO em C# de maneira prática e demonstrativa. Esperamos que este estudo tenha contribuído para uma compreensão mais sólida da Programação Orientada a Objetos e sua aplicação na linguagem C#. A POO é uma ferramenta poderosa para a criação de software de alta qualidade e sua compreensão é fundamental para qualquer desenvolvedor de software moderno.

Esta pesquisa serve como um ponto de partida para futuras explorações e aprendizados mais profundos no campo da programação orientada a objetos e na linguagem C#. Concluímos este

trabalho com a certeza de que a POO é uma abordagem essencial para o desenvolvimento de software confiável e eficiente, capaz de enfrentar os desafios da computação contemporânea.

Referências bibliográficas

https://blog.betrybe.com/tecnologia/poo-programacao-orientada-a-objetos/