AGENDA 14

UTILIZANDO MICROSOFT MAUI PARA DESENVOLVIMENTO MOBILE

GEEaD - Grupo de Estudos de Educação a Distância

Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO

EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO

CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO MOBILE I

Expediente

Autores:

TIAGO ANTONIO DA SILVA

KELLY CRISTIANE DE OLIVEIRA DAL POZZO

Revisão Técnica:

GUILHERME HENRIQUE GIROLLI

São Paulo – SP, 2024

Orientação a Objetos

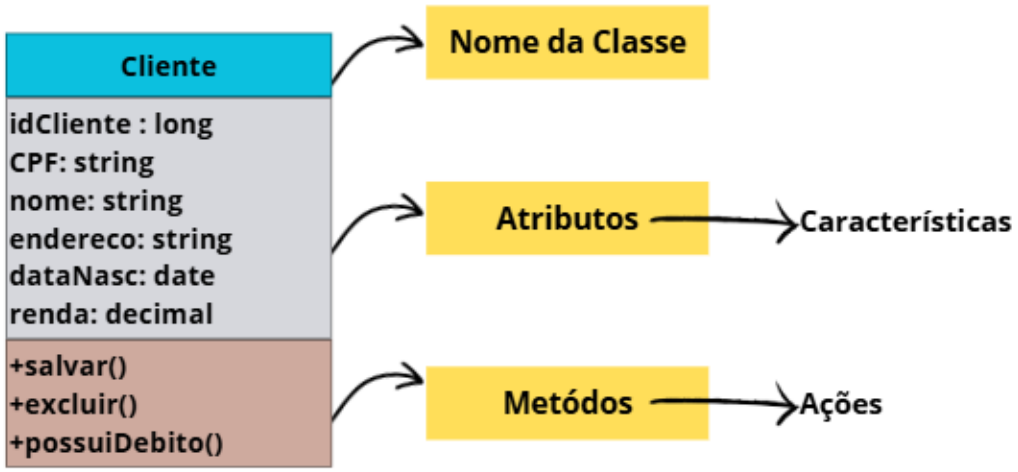
Vimos brevemente sobre o conceito de orientação a objetos primeiro módulo do curso, agora vamos relembrar sobre esse paradigma tão importante para a programação.

O QUE ORIENTAÇÃO A OBJETOS?

A orientação a objetos é um paradigma de programação que facilita a organização, manutenção e expansão do código, tornando-o mais modular, reutilizável e intuitivo. Esse paradigma organiza o código em torno de objetos, que representam entidades do mundo real (como uma pessoa ou um carro) ou conceitos abstratos (como uma conta bancária ou uma transação). Cada objeto é uma instância de uma classe e contém atributos (dados que descrevem seu estado) e métodos (comportamentos ou ações que ele pode realizar).
Com a orientação a objetos, o código é estruturado para refletir o comportamento e as interações entre os elementos que ele representa, permitindo ao desenvolvedor pensar em termos de objetos com características e funcionalidades específicas. Esse modelo promove maior modularidade, pois cada objeto é independente e autocontido, facilitando a reutilização e atualização do código em projetos de qualquer escala.

CLASSE

Uma classe é uma estrutura que serve como um modelo para criar objetos. Ela define as características (atributos) e comportamentos (métodos) que os objetos desse tipo terão. Em outras palavras, a classe descreve o que o objeto é e o que ele pode fazer, mas não representa um objeto específico até que seja instanciada.



Os principais conceitos da orientação a objetos são:

- ✓ **Classe e Objeto**: A classe é um modelo ou plano, enquanto o objeto é uma instância concreta dessa classe. Por exemplo, a classe Carro define o que é um carro, e cada carro específico (um Honda, um Toyota) é um objeto dessa classe.
- ✓ **Encapsulamento**: Refere-se ao ato de esconder detalhes internos de um objeto, expondo apenas o que é necessário. Isso melhora a segurança e a modularidade do código.
- ✓ **Herança**: Permite criar uma nova classe que herda características de uma classe existente, promovendo a reutilização de código. Por exemplo, a classe CarroEsportivo pode herdar de Carro.
- ✓ **Polimorfismo**: Permite que objetos de diferentes classes sejam tratados como instâncias de uma mesma classe base, permitindo comportamentos diferentes para métodos com o mesmo nome.
- ✓ **Abstração**: Consiste em focar apenas nos aspectos essenciais de um objeto, ignorando detalhes desnecessários, e modelá-los de forma a representar conceitos do mundo real no código.

Utilize ferramentas de Inteligência Artificial para pesquisar mais sobre o Paradigma Orientado a Objetos, pesquise sobre os principais conceitos e solicite exemplos práticos aplicados na linguagem C#.

Saiba mais sobre o conceito de Orientação a Objetos assistindo ao vídeo PROGRAMAÇÃO ORIENTADA A OBJETO (POO) - O que é? Entenda:

Disponível em: https://www.youtube.com/watch?v=f-aDDLRmugU. Acessado em: 31/10/2024.

MERGULHANDO NO TEMA

Chegou a hora de adicionar um aspecto mais dinâmico ao projeto AppHotel com o uso do .NET MAUI. Esta agenda abordará a implementação de código C# para carregar elementos da interface gráfica e realizar validações em componentes DatePicker. Além disso, será aprofundado o conhecimento em orientação a objetos em C#, incluindo o conceito de array de objetos.

Classes em C#

Uma classe em C# é um molde ou uma estrutura que define um tipo de objeto. Ela encapsula dados (propriedades) e comportamentos (métodos) que representam entidades do mundo real ou conceitos abstratos. As classes são a base da programação orientada a objetos e permitem a criação de objetos (instâncias) que compartilham a mesma estrutura e comportamento definidos pela classe.

Vamos a um exemplo de uma classe básica desenvolvida em C#:

```
public class Pessoa
{
    // Propriedades
    public string Nome { get; set; }
    public int Idade { get; set; }
    // Método
    public void Apresentar()
    {
        Console.WriteLine($"Olá, meu nome é {Nome} e tenho {Idade} anos.");
    }
}
```

Vamos analisar o exemplo apresentado:

- ✓ **Modificador de Acesso public:** Torna a classe acessível de qualquer parte do código.

- ✓ **Propriedades Nome e Idade:** Representam os dados da pessoa, as caracteristicas.
- ✓ **Método Apresentar:** Representa o comportamento da pessoa ao se apresentar, as ações.

Instanciando uma Classe – Criando um Objeto

Instanciar um objeto significa criar uma cópia concreta de uma classe na memória do programa, permitindo seu uso e manipulação, em C#, isso é feito com o operador **new**. Por exemplo, para utilizar a classe Pessoa, criamos uma instância (objeto) dela:

```
Pessoa pessoa1 = new Pessoa();
pessoa1.Nome = "Maria";
pessoa1.Idade = 30;
pessoa1.Apresentar(); // Saída: Olá, meu nome é Maria e tenho 30 anos.
```

Métodos Getter e Setter

Os métodos getter e setter são utilizados na orientação a objetos para acessar e modificar os valores de atributos privados de uma classe, promovendo o encapsulamento. O get permite obter o valor de um atributo, enquanto o set permite definir ou alterar esse valor. Esse controle ajuda a proteger os dados internos da classe, garantindo que os atributos sejam acessados de forma segura. Vamos detalhar mais sobre o assunto mais adiante.

Acesso as propriedades

As propriedades são membros da classe que fornecem um meio de ler, escrever ou calcular os valores dos campos privados. Elas são uma forma encapsulada de acessar os dados internos de uma classe. Em C# temos dois tipos de propriedades:

Auto-implementadas: Simplificam a declaração de propriedades sem a necessidade de campos privados explícitos.

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

Com Implementação Personalizada: Permitem lógica adicional nos métodos get e set.

```
private int _idade;

public int Idade
{
    get { return _idade; }
    set
    {
        if (value >= 0)
            _idade = value;
        else
            throw new ArgumentException("Idade não pode ser negativa.");
    }
}
```

Quando implementamos as propriedades também podemos controlar o acesso a elas usando modificadores, sendo:

- ✓ **public:** Acesso livre de qualquer parte do código.

- ✓ **private:** Acesso restrito apenas à própria classe.
- ✓ **protected:** Acesso permitido na classe e em classes derivadas.

Podemos definir propriedades que são apenas leitura ou apenas escrita.

- ✓ **Somente Leitura:**

```
public string Nome { get; }
```

- ✓ **Somente Escrita:**

```
public string Nome { private get; set; }
```

Vamos entender o set e get

Os métodos get e set são usados para acessar e modificar os valores das propriedades de uma classe. Eles permitem encapsular a lógica de acesso aos dados, oferecendo controle sobre como os valores são lidos ou alterados.

Vamos revisar o exemplo da classe Pessoa com uma implementação personalizada para a propriedade Idade:

```
 public class Pessoa
{
    private int _idade;

    public int Idade
    {
        get
        {
            return _idade;
        }
        set
        {
            if (value >= 0)
                _idade = value;
            else
                throw new ArgumentException("Idade não pode ser negativa.");
        }
    }
}
```

Como funciona?

get: Retorna o valor do campo privado _idade.

set: Define o valor do campo _idade, com uma validação para garantir que a idade não seja negativa.

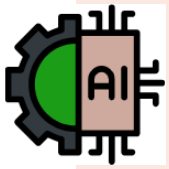
E a palavra-chave value?

A palavra-chave value é usada dentro do método set para representar o valor que está sendo atribuído à propriedade. Ela permite que você trabalhe com o valor que o usuário está tentando definir. Vamos ao exemplo:

```
public int Idade
{
    get { return _idade; }
```

```
    set
    {
        if (value >= 0)
            _idade = value;
        else
            throw new ArgumentException("Idade não pode ser negativa.");
    }
}
```

No exemplo apresentado o método **set** permite definir um novo valor para a **_idade**, mas inclui uma validação para garantir que a idade não seja negativa.
Se o valor **(value)** for maior ou igual a zero, **_idade** é atualizado com o valor.
Se o valor for negativo, o código lança uma exceção ArgumentException com e exibe a mensagem "**Idade não pode ser negativa.**". Essa exceção impede que valores inválidos sejam atribuídos, reforçando a integridade dos dados.



Utilize ferramentas de Inteligência Artificial e questione sobre o conceito de encapsulamento de dados na orientação a objetos. Para melhor compreensão sobre a aplicação em uma linguagem de programação, solicite um exemplo em C#.

Em nosso projeto AppHotel vamos implementar classes, na videoaula é possível visualizar os conceitos apresentados até aqui na prática.

Array de Objetos com Tamanho Dinâmico com List

Um array de objetos é uma estrutura de dados que armazena múltiplas instâncias de objetos do mesmo tipo em uma sequência ordenada. Em C#, arrays são coleções de tamanho fixo ou variável, onde cada elemento é acessado por um índice.

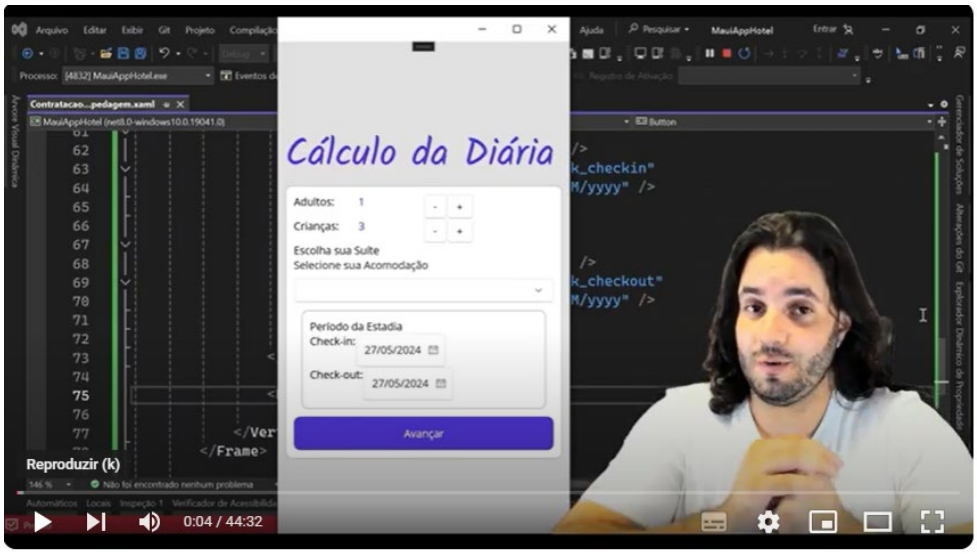
Mas porque utilizar Arrays de Objetos?

- ✓ **Organização de Dados:** Permitem agrupar objetos relacionados de forma estruturada.
- ✓ **Acesso Rápido:** Proporcionam acesso eficiente aos elementos através de índices.
- ✓ **Manipulação Simplificada:** Facilita operações como iteração, busca e ordenação.

Vamos a um exemplo de declaração de array de objetos usando List com a classe Pessoa:

```
List<Pessoa> listaPessoas = new List<Pessoa>();
listaPessoas.Add(new Pessoa("Maria", 30));
listaPessoas.Add(new Pessoa("João", 25));
listaPessoas.Add(new Pessoa("Ana", 28)); // Fácil adição de novos elementos
```

Assista o vídeo apresentando a continuação do desenvolvimento do AppHotel, onde foi aplicado os conceitos apresentados nesta agenda:



Disponível em: https://www.youtube.com/watch?v=XJSnY5bBm9s

CÓDIGO-FONTE DESTA AGENDA:

https://github.com/tiagotas/MauiAppHotel/tree/4e655cb68610572865b83fc420655601b1a5f1ff