**Linguagem de programação C# Orientada a Objetos**

**Lucas Matheus Casarotti Rodrigues**

**Sumário**

[1. Introdução 3](#_Toc145507300)

[1.1. O que é programação orientada a objetos (POO)? 3](#_Toc145507301)

[1.2. Vantagem programação orientado a objeto 3](#_Toc145507302)

[1.2.1. Modelagem do Mundo Real 3](#_Toc145507303)

[1.2.2. Encapsulamento 3](#_Toc145507304)

[1.2.3. Reutilização de Código 3](#_Toc145507305)

[1.2.4. Polimorfismo 3](#_Toc145507306)

[1.2.5. Abstração 3](#_Toc145507307)

[2. C# POO fundamental 3](#_Toc145507308)

[2.1. Classes 3](#_Toc145507309)

[2.2. Atributos 4](#_Toc145507310)

[2.3. Métodos 4](#_Toc145507311)

[2.4. Objetos 4](#_Toc145507312)

[2.5. Métodos com Return 5](#_Toc145507313)

[2.5.1. Quando utilizar cada um? 5](#_Toc145507314)

[2.6. Modificadores de acesso 6](#_Toc145507315)

[2.7. Constructor 6](#_Toc145507316)

[2.8. Encapsulamento 7](#_Toc145507317)

[2.9. Get e Set 8](#_Toc145507318)

[2.10. This 9](#_Toc145507319)

[2.11. Herança 10](#_Toc145507320)

[2.12. Polimorfismo 11](#_Toc145507321)

[2.13. Abstração 12](#_Toc145507322)

[2.13.1. Classes Abstratas 12](#_Toc145507323)

[2.13.2. Métodos abstratos 12](#_Toc145507324)

[2.14. Interface 13](#_Toc145507325)

[2.15. Classe estáticas / Membros estáticos 14](#_Toc145507326)

[2.16. Classe Selada 15](#_Toc145507327)

[2.17. Classe Partial 15](#_Toc145507328)

[2.18. Comparando objetos com Equals 16](#_Toc145507329)

[2.19. Listas 16](#_Toc145507330)

# Introdução

## O que é programação orientada a objetos (POO)?

Resumidamente, a programação orientada a objetos é um modelo de programação que tem como objetivo representar objetos do mundo real. Para isso, são utilizadas classes (modelos), atributos (características) e métodos (funcionalidades).

Exemplo:

Comprei um carro (classe carro) da cor azul (atributo) e esse carro acelera, desacelera e buzina (métodos).

## Vantagem programação orientado a objeto

A orientação a objetos (OO) é um paradigma de programação que utiliza conceitos de objetos e classes para modelar o mundo real e organizar o código de maneira mais eficaz. Aqui está um resumo das vantagens da programação orientada a objetos:

### Modelagem do Mundo Real

A OO permite que você modele objetos e suas interações de forma semelhante ao mundo real, tornando mais fácil entender e representar sistemas complexos.

### Encapsulamento

Os objetos encapsulam seus dados e funcionalidades, controlando o acesso a eles. Isso ajuda a proteger os dados e promove a ocultação de informações.

### Reutilização de Código

A herança e a composição permitem a reutilização de código, economizando tempo e esforço de desenvolvimento.

### Polimorfismo

O polimorfismo permite que objetos de classes diferentes se comportem de maneira uniforme quando expostos à mesma interface. Isso promove a flexibilidade e a extensibilidade do código.

### Abstração

A OO permite a criação de abstrações de objetos e classes, simplificando a complexidade do sistema e focando nos aspectos essenciais.

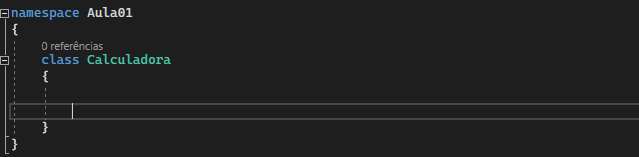
# C# POO fundamental

## Classes

É um modelo de um objeto do mundo real.

Exemplo:

Criamos uma classe chamada ‘Calculadora’, essa classe serve para representar uma calculadora do mundo real, nessa classe pode conter atributos (características) e métodos (funcionalidades).

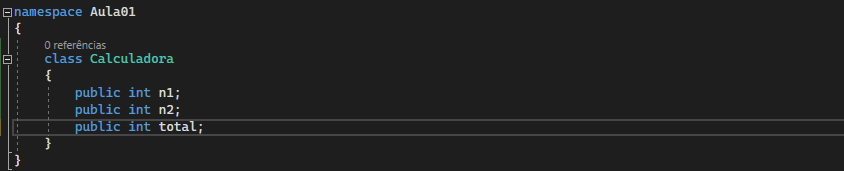


## Atributos

Os atributos em programação orientada a objetos são utilizados para representar as características da classe.

Exemplo:

Criamos uma classe do tipo pessoa que tem duas características.

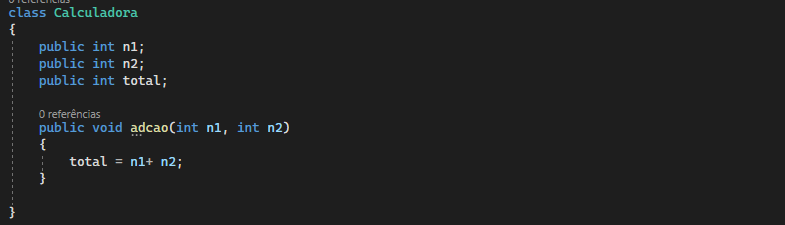


## Métodos

É as funcionalidades que uma classe contém, como (cadastrar, excluir, alterar).

Exemplo:

Primeiro foi criado uma classe com o nome de calculadora, com três atributos e um método do tipo void, ou seja, que não tem retorno ele apenas realiza a adição dos números e atribui ao atributo total.

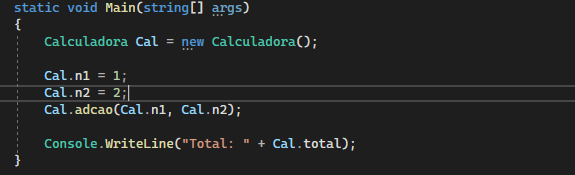


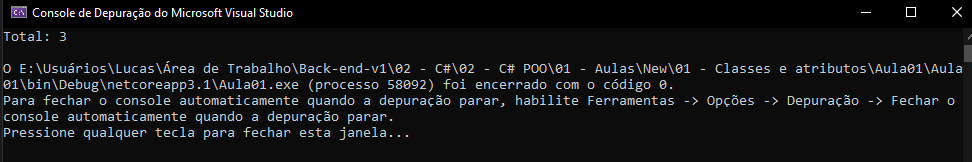
## Objetos

É criado a partir da classe, ou seja, é uma instância específica de uma classe. Ele representa uma entidade ou conceito concreto do mundo real e possui características e comportamentos definidos pela classe da qual é instanciado.

Exemplo:

Foi criado um objeto chamado ‘Cal’ que é do tipo calculadora, depois foi adicionado valores a seus atributos e logo em seguida chamado o método ‘adcao’ que realizada o cálculo dos dois atributos.

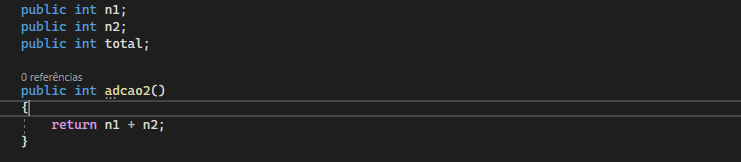


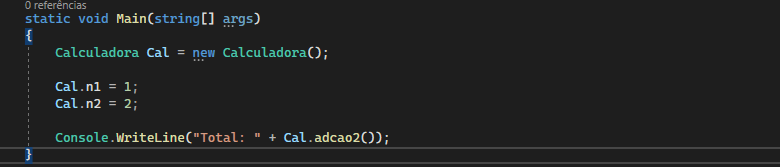


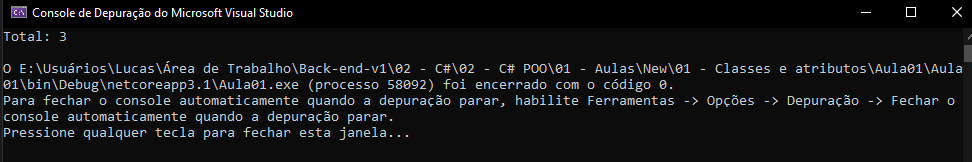
## Métodos com Return

Diferente do método do tipo void, ele não recebe parâmetros de entrada, apenas retorna a soma dos dois números, sem a necessidade de criar outro atributo para receber os valores n1 e n2.

Exemplo:







### 2.5.1. Quando utilizar cada um?

O método do tipo void é usado para quando queremos salvar algum valor em uma variável. Já o método do tipo return, como o próprio nome já diz, ele é utilizando quando já queremos retornar algo sem a necessidade de salvar em alguma variável.

## Modificadores de acesso

São as palavras-chaves usadas para especificar a acessibilidade de uma classe, método e atributos.

public = Atributos e métodos visíveis para todas as classes.

private = Atributos e métodos visíveis apenas onde foi criado.

protected = Atributos e métodos visíveis em classes onde são criados ou herdados.

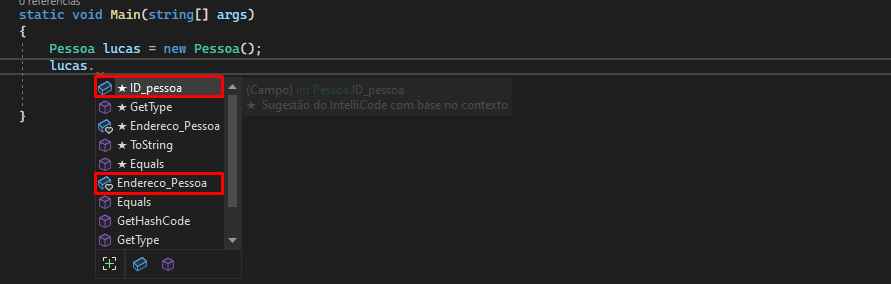
Internal = Atributos e métodos visíveis apenas para o mesmo namespace onde foi criado.

Exemplo:

Foi criado uma classe do tipo pessoa com alguns atributos, cada um com um tipo diferente de modificador de acesso.



após instanciar um objeto do tipo pessoa na classe main, observe que apenas os atributos do tipo public e internal são visíveis para o objeto criado.

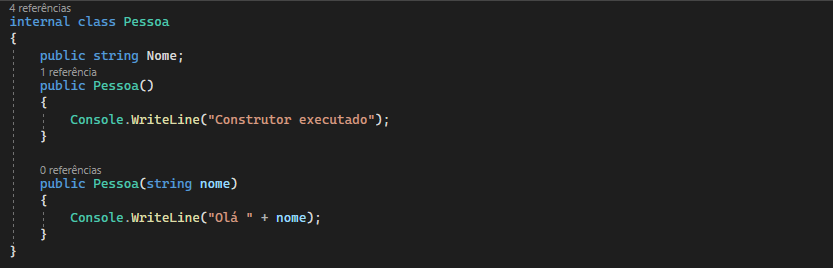


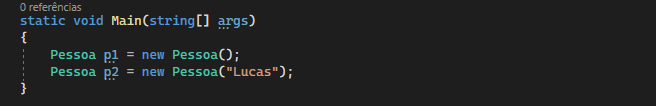
## Constructor

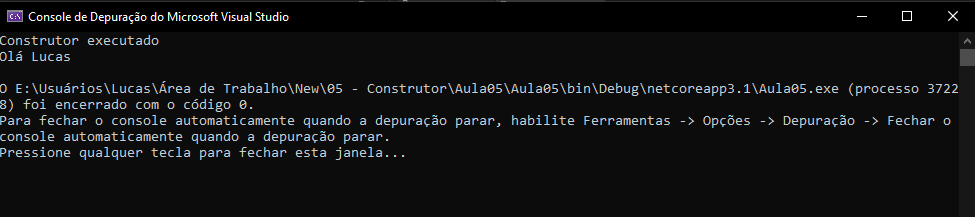
Um construtor (constructor) é um método especial de uma classe que é executado automaticamente quando um objeto daquela classe é criado. Ele é responsável por inicializar os atributos do objeto e realizar qualquer outra configuração inicial necessária.

Exemplo:

Foi criado dois construtores para a classe ‘Pessoa’, podemos criar vários construtores para a mesma classe desde que ele tenha os parâmetros de entrada diferentes, esses parametros são passados diretamente na hora de criarmos o objeto.



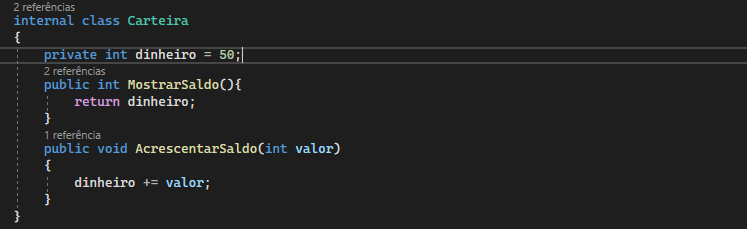


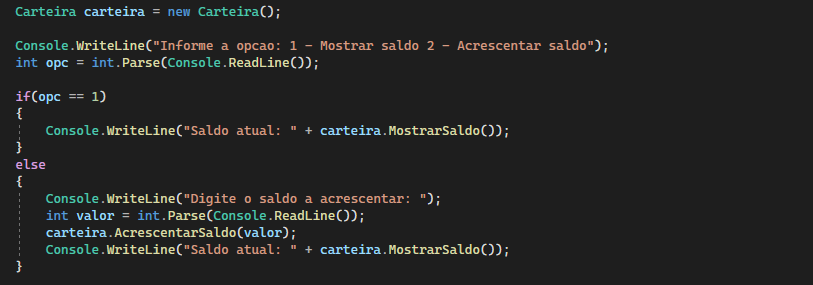
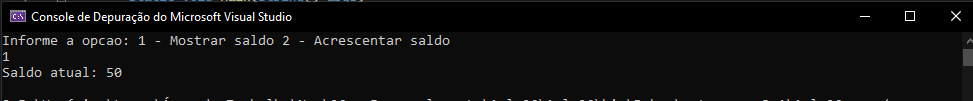


## Encapsulamento

O encapsulamento é o processo de ocultar ou esconder os membros de uma classe do acesso exterior usando modificadores de acesso. O encapsulamento também é chamado de ocultação de informação. O encapsulamento fornece uma maneira de preservar a integridade do estado dos dados.

Exemplo:

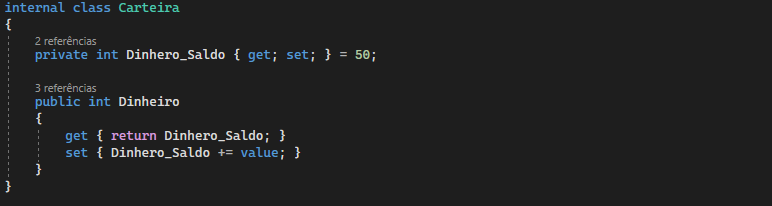
Foi criada uma classe chamada 'Carteira' que contém um atributo chamado 'dinheiro'. Para evitar que o usuário tenha acesso direto a esse atributo, foram utilizados modificadores de acesso do tipo 'private'. A única maneira de acessar esses atributos é utilizando os métodos 'MostrarSaldo' e 'AcrescentarValor'.

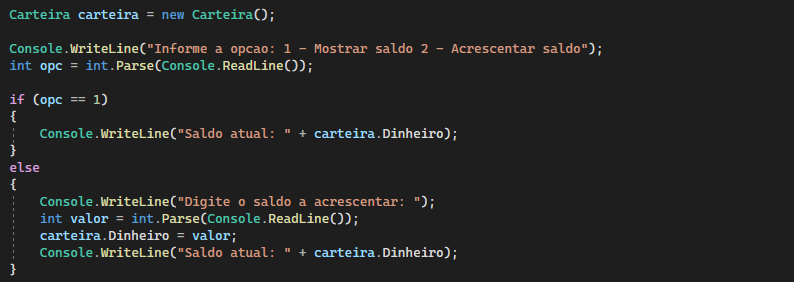


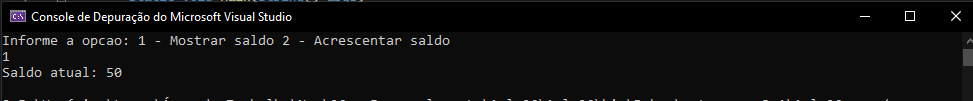
## Get e Set

Os conceitos de Get e Set são métodos utilizados para acessar (Get) e modificar (Set) os valores dos atributos de uma classe. Esses métodos fazem parte do conceito de encapsulamento e são usados para controlar o acesso aos atributos, permitindo que isso seja feito de forma controlada e segura. Os métodos Get e Set podem ser implementados através de propriedades, uma propriedade é uma espécie de método que representa um atributo da classe.

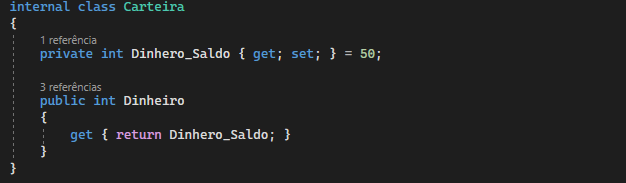
Exemplo:

Neste exemplo, utilizaremos a classe chamada 'Carteira', mas, em vez de criar dois métodos para alterar e mostrar o saldo, usaremos os métodos get e set da propriedade.





Também é possível definir níveis de acesso para os métodos Get e Set, tornando certos atributos somente leitura (apenas com Get) ou somente escrita (apenas com Set).

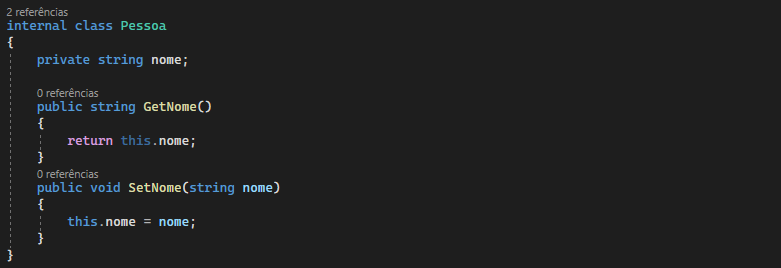


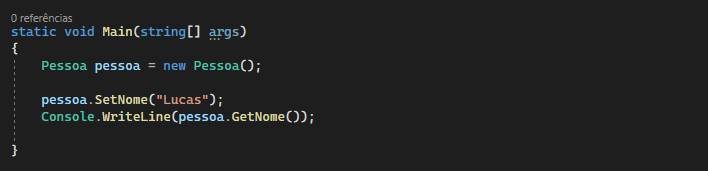
## This

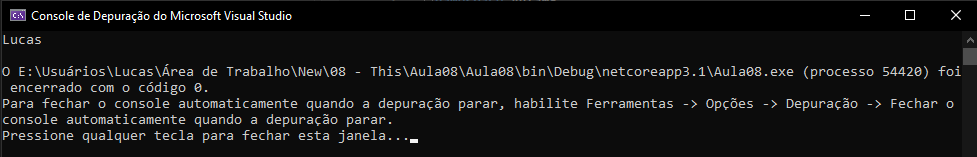
A palavra-chave "this" é utilizada para se referir ao próprio objeto da classe em que está sendo utilizada. Ela é usada para diferenciar entre membros da instância atual da classe e parâmetros de método ou variáveis locais que têm o mesmo nome.

Exemplo:

Foi criada uma classe chamada 'Pessoa' com o atributo 'nome' e dois métodos. Um dos métodos retorna o nome atual da pessoa, e o outro modifica o atributo 'nome'. Para referenciar o atributo dentro dos métodos e facilitar a identificação, usamos 'this.nome'. Dessa forma, sabemos qual é o objeto e qual é o parâmetro de entrada, pois ambos têm o mesmo nome.







* 1. Herança

A herança é um dos principais conceitos da programação orientada a objetos, ela permite que uma classe (chamada classe derivada ou sub classe) herde atributos e métodos de outra classe (chamada classe base ou super classe).

Exemplo:

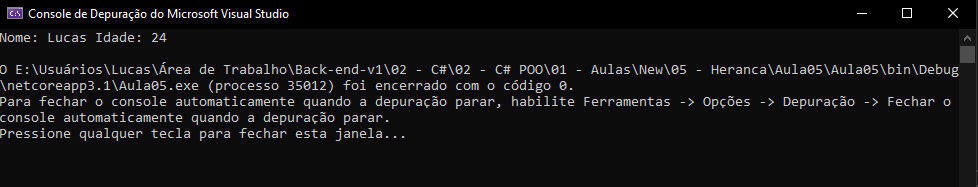
Criamos uma classe chamada 'Pessoa' com dois atributos, nome e idade, e um método.

Logo em seguida, criamos uma classe chamada 'Funcionário' com os mesmos atributos da classe 'Pessoa'.

Ao invés de criar os mesmos atributos para ambas as classes, podemos definir que a classe Funcionário herda da classe Pessoa, desta maneira criamos uma hierarquia.

Na classe principal quando criarmos um objeto do tipo funcionário esse mesmo objeto terá acesso a todos atributos e métodos da classe pessoa, pois a classe Funcionário herda da classe Pessoa.



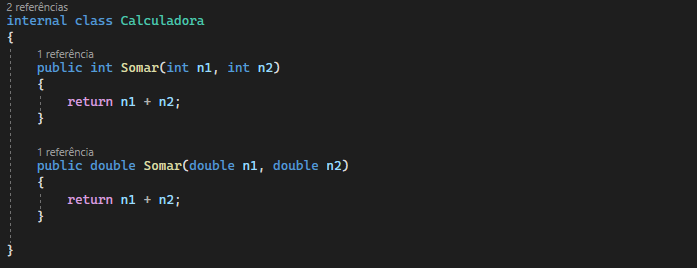


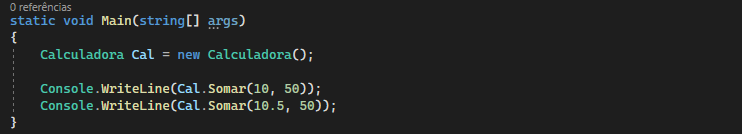
## Polimorfismo

O polimorfismo permite que um objeto possa ser trato de várias formas diferentes, ou seja, um mesmo método pode ser utilizado para objetos de classes distintas, desde que essas classes herdem de uma mesma classe base. Existem duas maneiras de implementar o polimorfismo através do polimorfismo de sobrecarga (métodos com o mesmo nome, mas com parâmetros diferentes) e do polimorfismo de sobreposição (métodos com a mesma assinatura em classes diferentes).

Exemplo de Polimorfismo de Sobrecarga:

Neste exemplo, a classe Calculadora possui dois métodos somar, um que realiza a soma de dois inteiros e outro que realiza a soma de dois números de ponto flutuante (double). Dependendo do tipo dos argumentos passados, o compilador selecionará a versão correta do método somar.

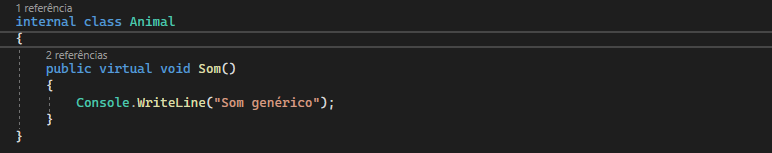




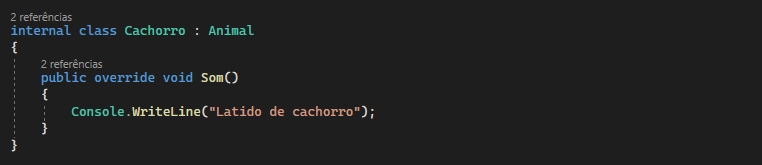


Exemplo de Polimorfismo de Sobreposição:

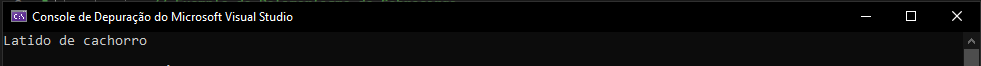
Neste exemplo, a classe animal possui um método virtual Som, que é sobreposto nas classes que herdarem da classe animal, ou seja, todo método do tipo virtual pode ser reescrito pelas classes que herdam da classe principal.



Para isso, foi criado uma classe chamada “Cachorro” que herda da classe animal, logo em seguida foi criado um método do tipo override Som, é o mesmo método da classe Animal, porém ele foi sobreposto de acordo com o que a classe precisa.







## Abstração

Métodos e classes abstratas são elementos fundamentais da Programação Orientada a Objetos (POO) que contribuem para a implementação da abstração, permitindo criar uma hierarquia de classes com comportamentos específicos e garantindo a consistência do código.

### Classes Abstratas

Uma classe abstrata é uma classe que não pode ser instanciada diretamente, mas pode ser utilizada como um modelo para outras classes derivadas. Ela pode conter métodos abstratos e métodos concretos (com implementação). Para indicar que uma classe é abstrata, utiliza-se a palavra-chave abstract.

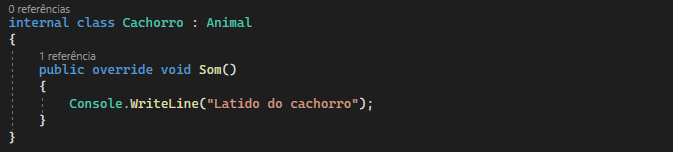
### Métodos abstratos

Um método abstrato é um método declarado em uma classe, mas não possui uma implementação. Ele é utilizado para representar um comportamento que deve ser obrigatoriamente implementado pelas classes derivadas (subclasses). A declaração de um método abstrato é precedida pela palavra-chave abstract, e ele não pode ter um corpo (ou seja, não possui chaves e código dentro).

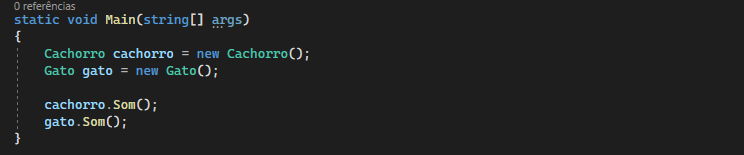
Exemplo:

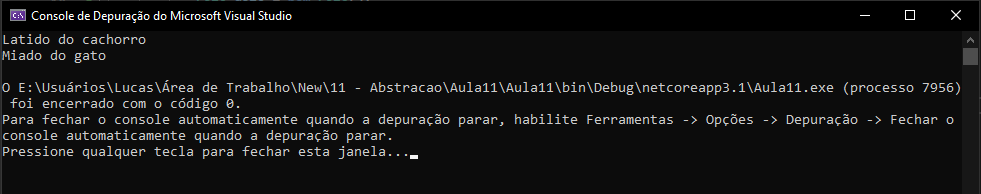
Neste exemplo, a classe abstrata “Animal” possui um método abstrato chamado “Som”. As classes “Cachorro” e “Gato” herdam da classe Animal, então são obrigadas a implementar o método Som e reescrever de acordo com a sua classe.











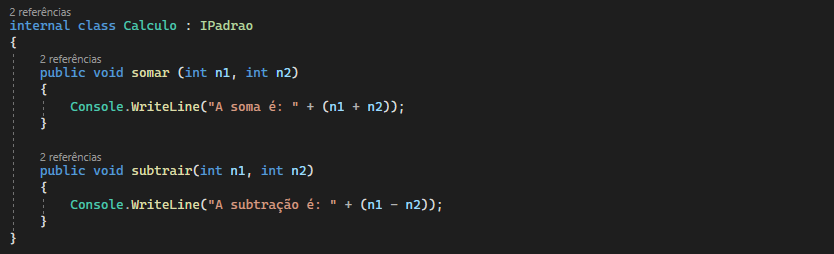
## Interface

Uma interface é um tipo de referência que define um conjunto de membros (métodos, propriedades, eventos e indexadores) que uma classe deve implementar. Ela é usada para definir um contrato ou um conjunto de comportamentos que uma classe deve ter para ser considerada compatível com essa interface. As interfaces permitem a implementação de polimorfismo e a separação clara entre a definição dos métodos e a sua implementação nas classes concretas.

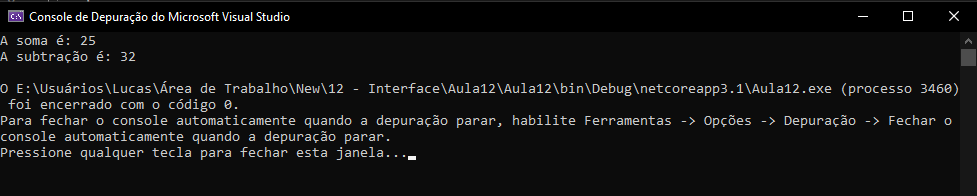
Exemplo:

Foi criada uma interface chamada 'IPadrao' com dois métodos: um para somar e outro para subtrair. Essa interface será herdada pela classe 'Calculo', e a classe 'Calculo' será obrigada a implementar esses métodos.







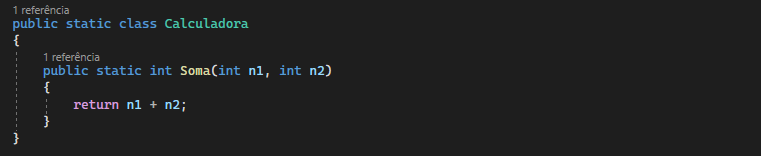


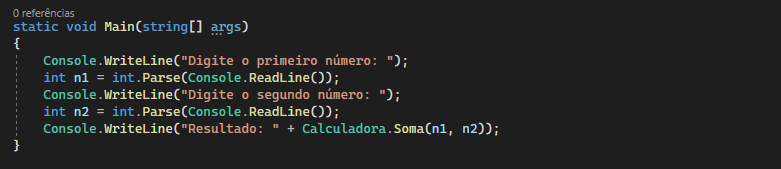
## Classe estáticas / Membros estáticos

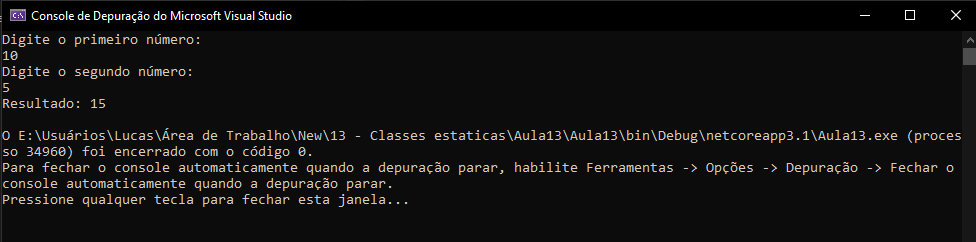
Uma classe estática é uma classe que contém apenas membros estáticos e não pode ser instanciada. Os membros estáticos são aqueles que pertencem à própria classe, em vez de pertencerem a instâncias individuais da classe. Eles são compartilhados por todas as instâncias da classe e podem ser acessados sem a necessidade de criar uma instância da classe.

Exemplo:

Foi criada uma classe chamada 'Calculadora' com um método para realizar a soma de dois números inteiros. Observou-se que, na classe principal, não instanciamos a classe 'Calculadora', mesmo assim conseguimos executar o método 'Soma'.



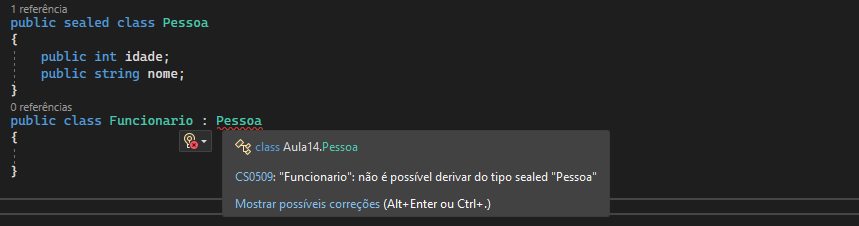




## Classe Selada

Resumidamente uma classe selada é para indicar que a classe não poderá ser herdada por outras.

Exemplo:



## Classe Partial

É um modificador de acesso que permite dividir a mesma classe em mais de um arquivo .cs, lembrando que a classe deve estar dentro do mesmo namespace. Geralmente é utilizado em classes muito grandes, é uma maneira de facilitar a manutenção do código.

Exemplo:

Foram criados dois arquivos .cs, um chamado ‘Pessoa’ e outro “Funcionario’. Em ambas tem a classe ‘Pessoa’ com seu modificador de acesso do tipo partial, porém com atributos diferentes. Desta forma apenas instanciando a classe pessoa podemos ter acesso aos dois atributos, mesmo existindo em arquivos .cs diferentes.





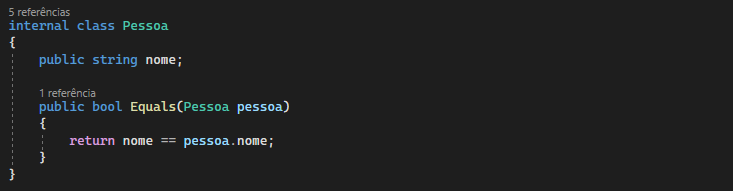


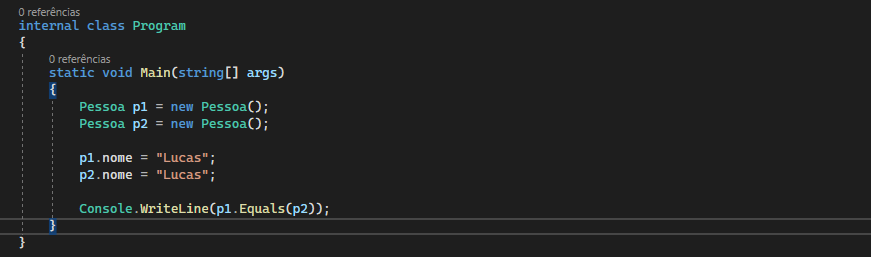
## Comparando objetos com Equals

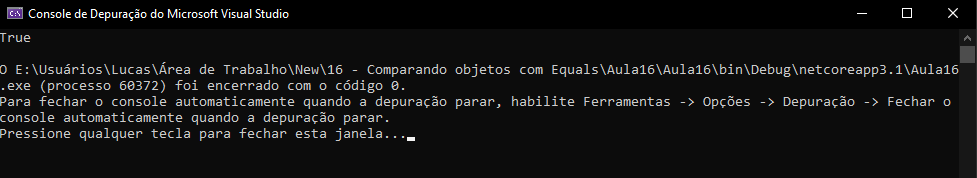
É um operador da classe Object utilizado para comparar se um objeto é igual ao outro.

Exemplo

Foi criado uma classe chamada ‘Pessoa’ com o atributo ‘nome’ e um método Equals, se os objetos forem iguais retorna true.







## Listas

No C# é possível criar listas de todos os tipos de dados, para isso é utilizado List, IEnumerable, entre outros.

Exemplo:

Foi criado uma classe chamada ‘Pessoa’, com dois atributos e um método construtor.



Na classe principal é necessário instanciar a classe pessoa do tipo List, desta maneira podemos listar todos os objetos do tipo pessoa.



