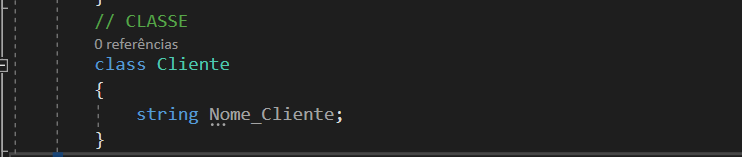
**C# Orientado a objetos**

**Programação Orientada a Objetos**

Programação Orientada a Objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos'.

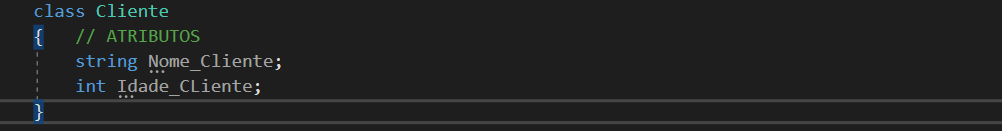
**Classes**

É uma descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos.



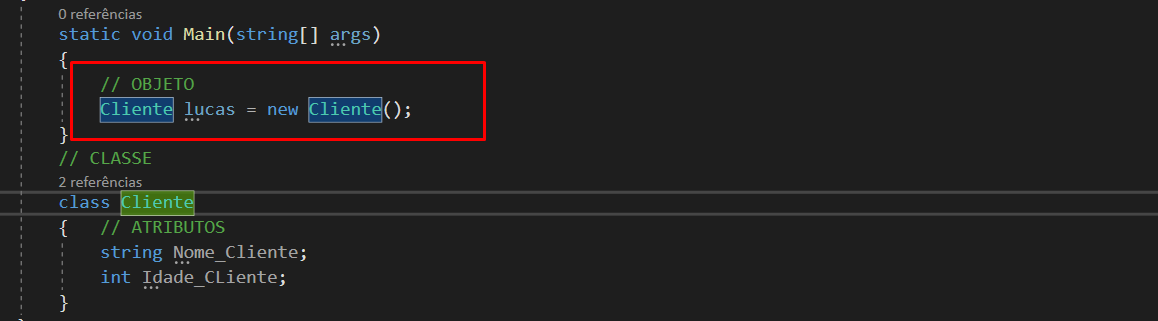
**Atributos**

Os Atributos em Programação Orientada a Objetos são os elementos que definem a estrutura de uma classe. Os atributos também são conhecidos como variáveis de classe, e podem ser divididos em dois tipos básicos: atributos de instância e de classe. Os valores dos atributos de instância determinam o estado de cada objeto.



**Objetos**

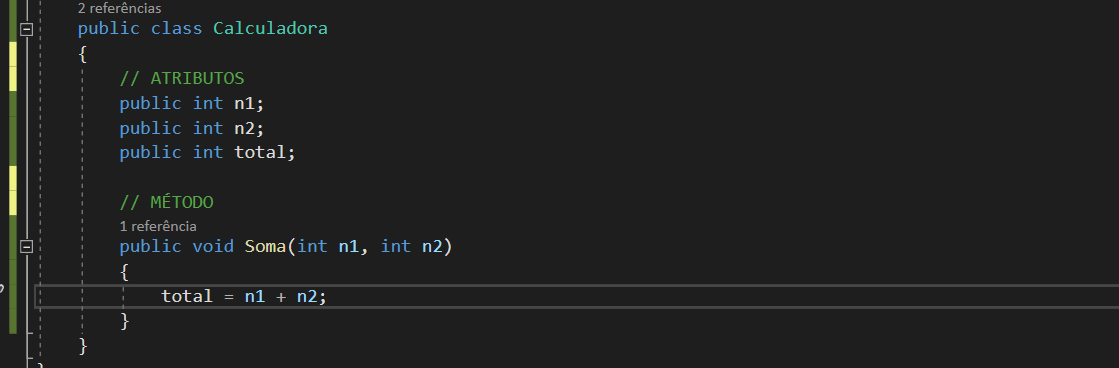
Ele é composto por propriedades, métodos e eventos da classe. Um objeto sempre será do tipo de referência, ou seja, ele só armazena o endereço dos dados.



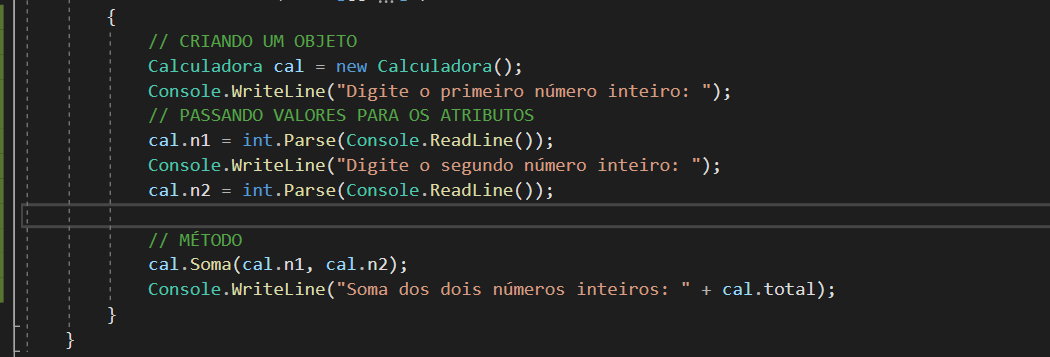
**Métodos**

É uma função é uma sub-rotina (ou procedimento ou função) associada a um objeto, e que possui acesso aos seus dados, as variáveis membro. Ele é executado por um objeto ao receber uma mensagem, como (cadastrar, excluir, alterar).

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.

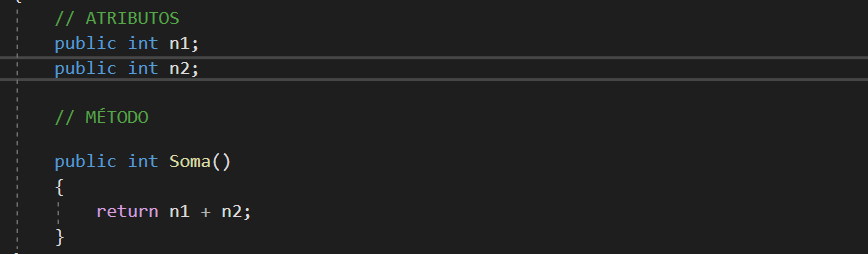


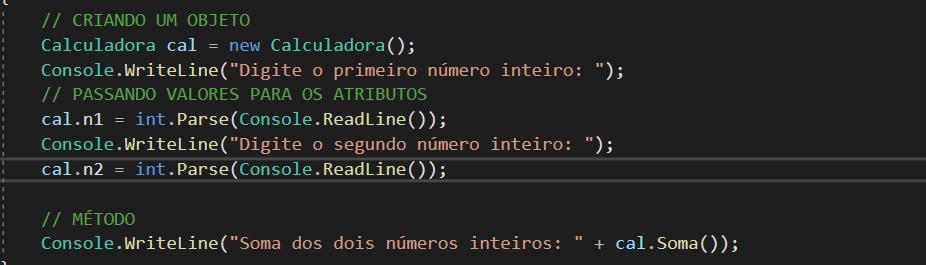
Em seguida foi criado um objeto, que é composto pelos atributos e métodos da classe calculadora



**Utilizando o mesmo método 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 outra variável para receber os valores n1 e n2.





**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.

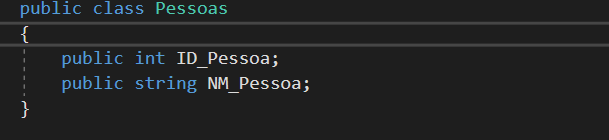
**Propriedades**

Uma propriedade (Property) é um membro de uma classe que fornece um mecanismo flexível para ler, gravar ou calcular o valor de um dado em particular.

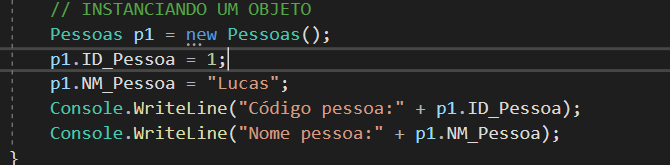
As propriedades permitem que uma classe exponha de uma maneira pública a obtenção e definição destes valores.

Exemplo:

Criamos uma classe do tipo pessoa com duas propriedades



desta forma em outra classe .cs podemos instanciar um objeto do tipo pessoa para acessar e atribuir valores a essas propriedades.



**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.

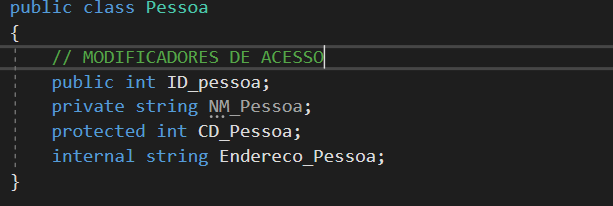
private = Atributos e métodos visíveis apenas onde foi criado.

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

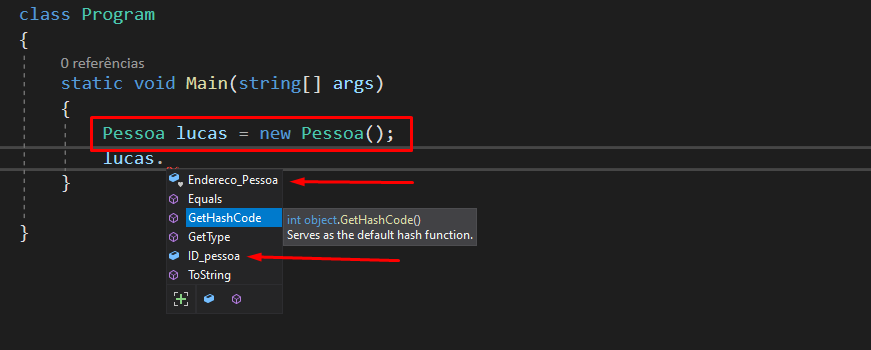
Internal = Atributos e métodos visíveis apenas para o mesmo namespace onde foi criado.

Exemplo:

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



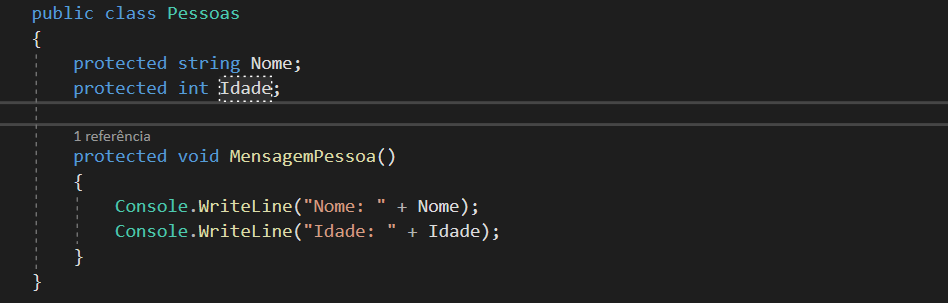
Agora iremos instanciar um objeto do tipo pessoa na classe main, observe que apenas os atributos do tipo public e internal são visíveis para o objeto criado.



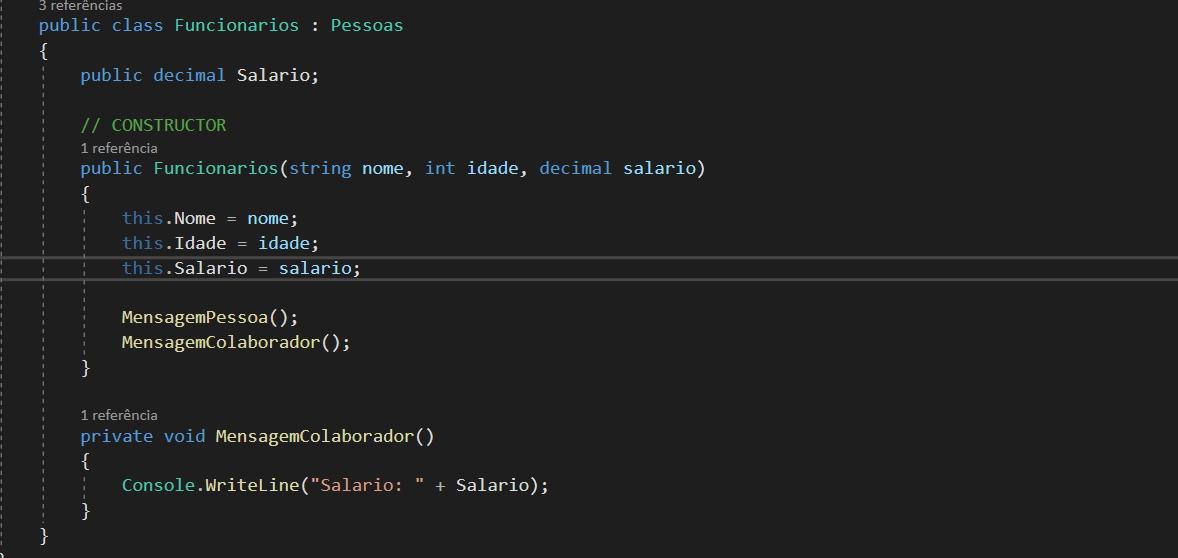
**Herança**

A herança é um princípio próprio à programação orientada a objetos (POO) que permite criar uma nova classe a partir de uma já existente. A principal vantagem da herança é a capacidade para definir novos atributos e métodos para a subclasse, que se somam aos atributos e métodos herdados.

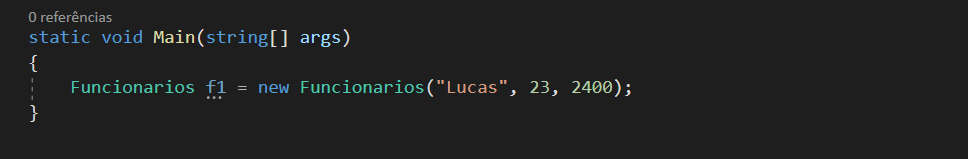
Exemplo: foi criado uma classe pessoa como modelo para as outras classes, com dois atributos e um método que imprime seu nome e idade.



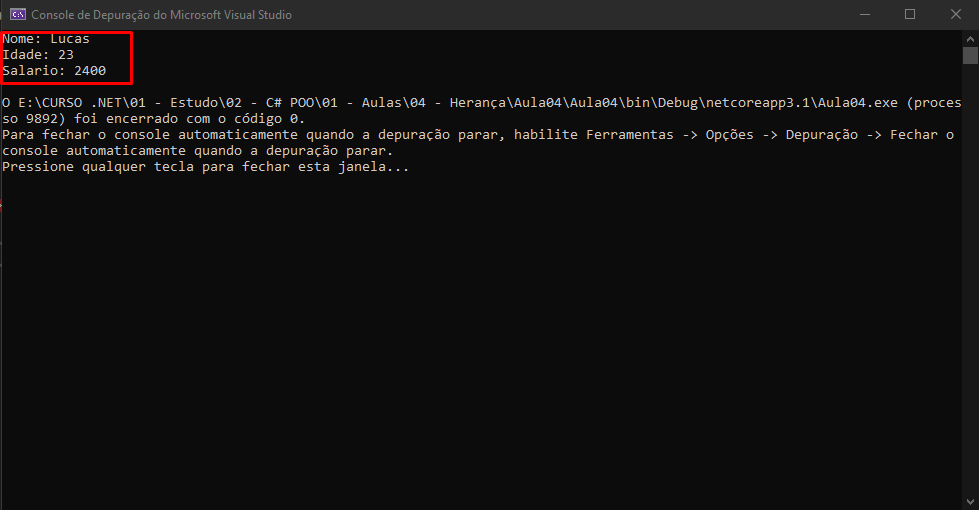
Em seguida foi criado uma segunda classe chamada Funcionários que herda da classe pessoas, ou seja, ela também tem os atributos e métodos criados na classe pessoas.



Na imagem acima foi utilizado um método do tipo construtor para iniciar os atributos com valores, que serão passados quando o objeto é instanciado.

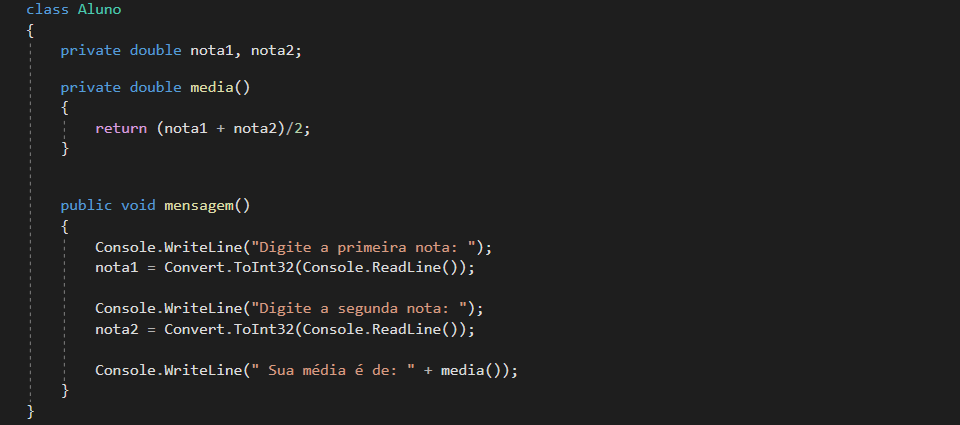


Instanciando um objeto já com valores nos seus atributos, no resultado podemos ver que é impresso o nome e idade, sendo da classe pessoas mesmo que criamos um objeto da classe funcionários, pois a classe funcionários consequentemente herda todos os atributos e métodos da classe pessoas.



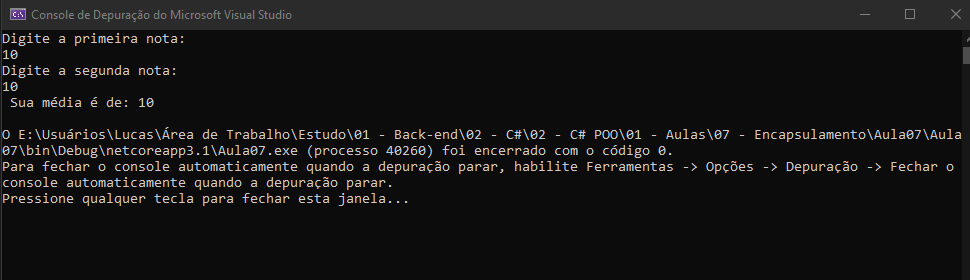
**Encapsulamento**

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 ou information hiding. O encapsulamento fornece uma maneira de preservar a integridade do estado dos dados.



Desta forma o usuário não tem acesso direto aos atributos da classe, a maneira para acessa-los é utilizando o método “mensagem”.





**Abstração**

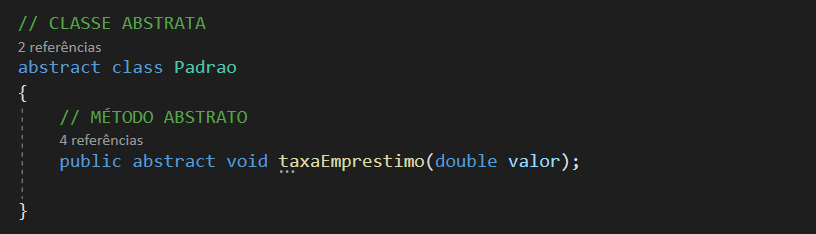
**Classes Abstratas**

Uma classe abstrata é uma classe que serve de modelo para outras classes. Ela não pode ser instanciada, apenas herdada pelas subclasses e pode conter ou não métodos abstratos, podendo ser implementados nas classes descendentes.

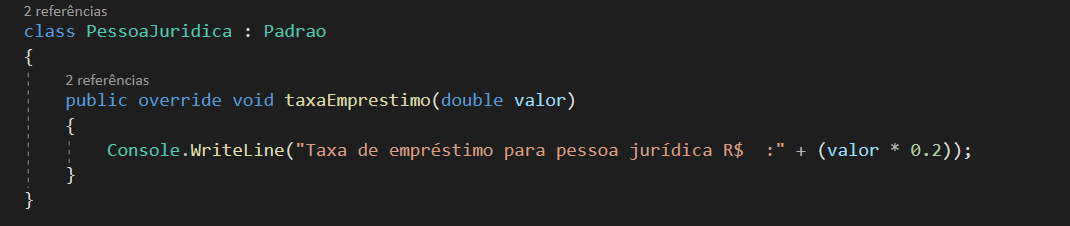
**Métodos abstratos**

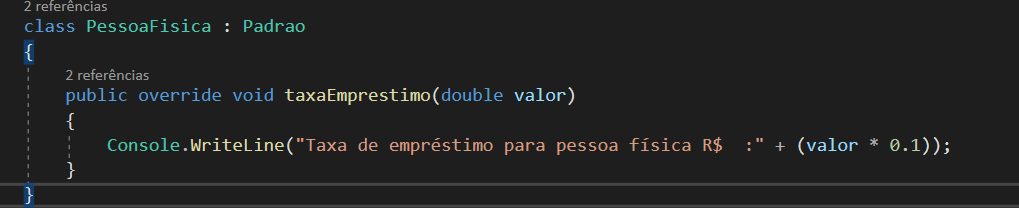
Métodos abstratos o corpo não tem implementação, ele é um método vazio que é implementado nas subclasses.

Exemplo: foi criado uma classe e um método abstrato para calcular a taxa de empréstimos para pessoas físicas e jurídicas.



Observe que nas duas imagens abaixo ambas herdam da classe Padrão, consequentemente tendo os métodos da classe herdada, porém o método herdado é do tipo abstrato, por padrão ele é criado sem implementação para ser implementados nas subclasses.



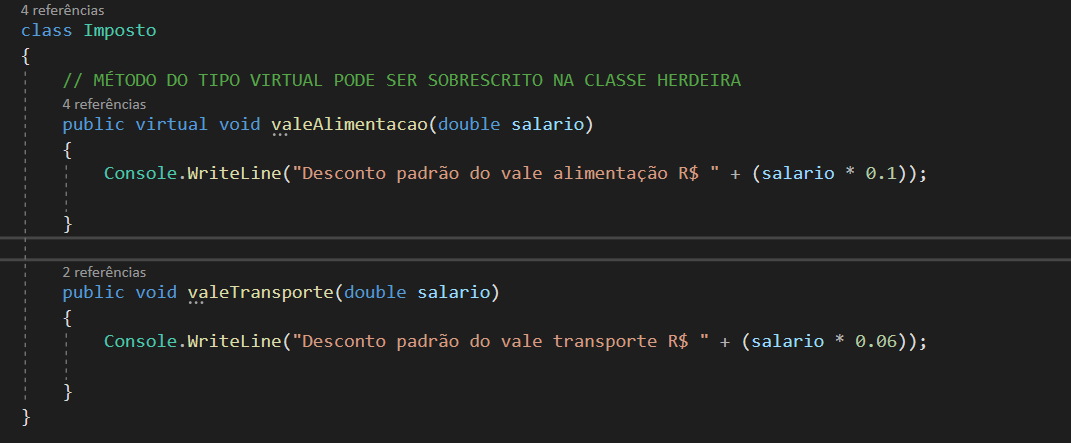


Ambas as subclasses tem métodos para calcular a taxa de empréstimos desta forma está sendo reescrito o método abstrato da classe abstrata.

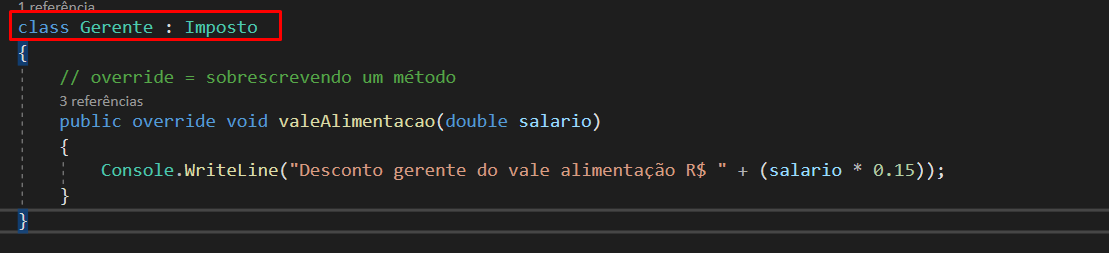
**Polimorfismo**

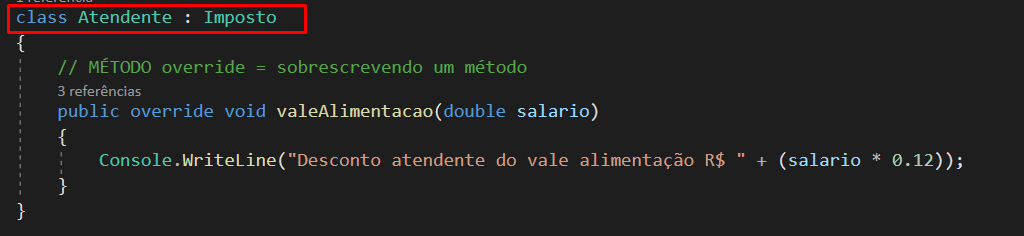
O Polimorfismo é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução. Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

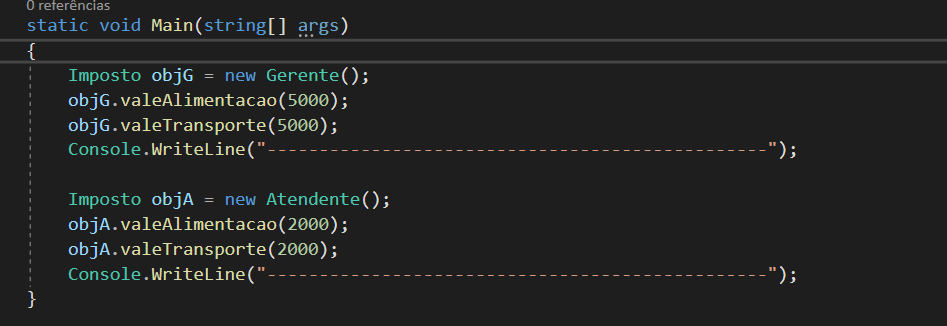
Exemplo: foi criado uma classe imposto para mostrar o desconto do vale alimentação e do vale transporte.



As duas imagens abaixo ambas herdam da classe imposto, contentando os seus métodos da classe herdada, diferente dos métodos abstratos mostrados anteriormente. Os métodos da classe Imposto já são implementados no momento da criação da classe, porém podem ser alterados nas subclasses.

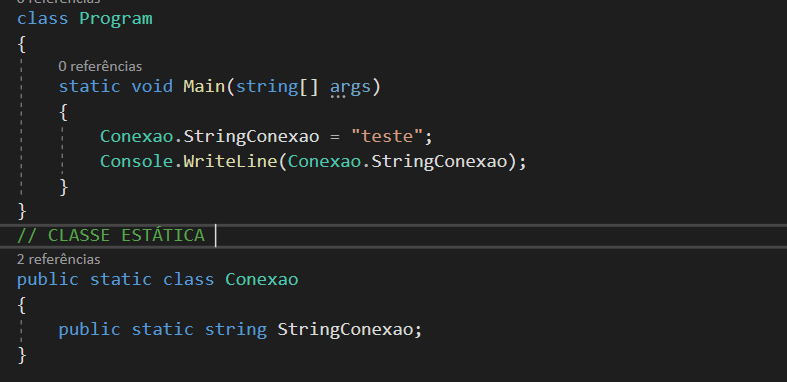






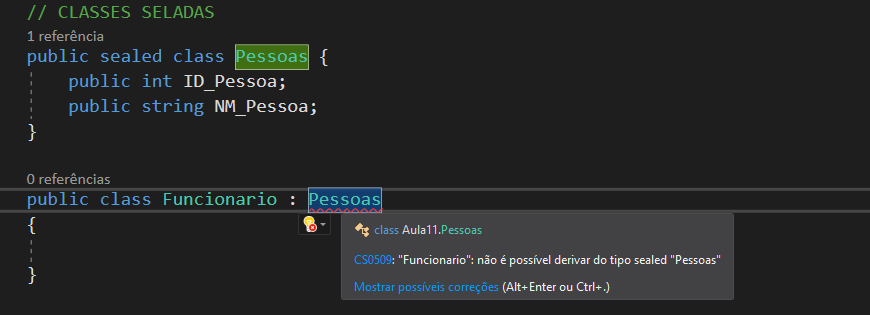
**Classe estáticas / Membros estáticos**

Membros estáticos são membros de dados (variáveis) ou métodos que pertencem a uma classe estática ou não estática, em vez de objetos da classe. Os membros estáticos sempre permanecem os mesmos, independentemente de onde e como são usados. Como os membros estáticos estão associados à classe, não é necessário criar uma instância dessa classe para invocá-los.



**Classe Selada**

Em resumo o sealed é para indicar que a classe não poderá ser herdada por outras.

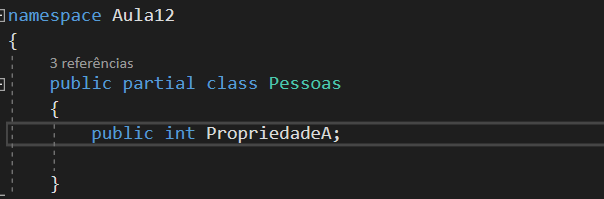


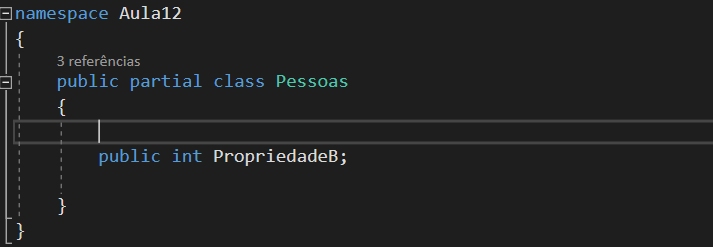
**Classe Partial**

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.

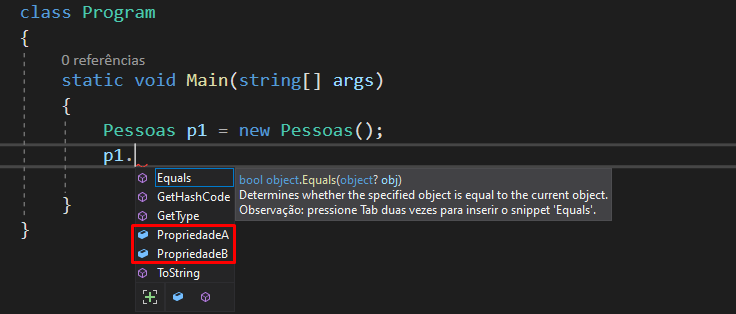
Exemplo:

Foi criado duas classes chamadas de Pessoas e outra de Funcionários





Ambas com uma classe com o mesmo nome, porém seu modificador de acesso é do tipo partial.

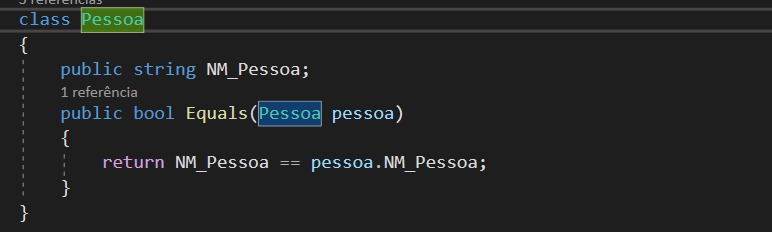


Na imagem acima podemos ver que mesmo instanciando apenas uma classe do tipo Pessoas podemos ter acesso as propriedades e métodos da classe Funcionário, pois ambas têm a mesma classe do tipo parcial, ou seja, foi divida em duas classes diferentes.

**Comparando objetos com Equals**

É um operador da classe Object utilizadas para comparar se um objeto é igual o outro.

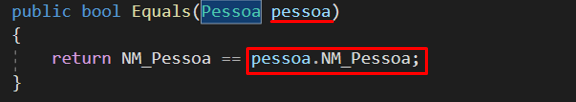
Exemplo: Foi criado uma classe do tipo pessoa com apenas uma propriedade NM\_Pessoa e um método Equals.



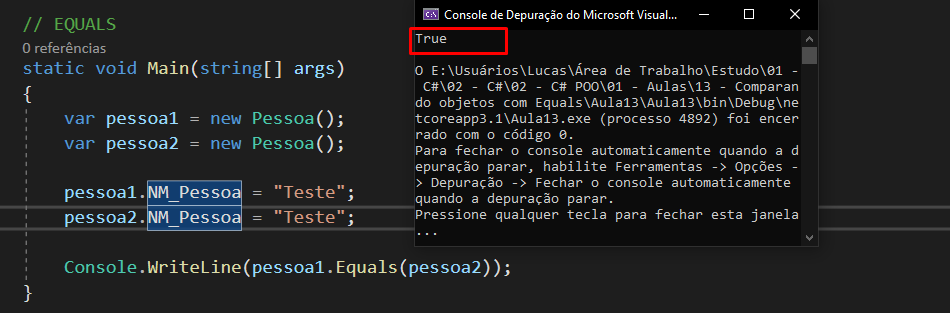
Na classe Main esse método é chamado da seguinte maneira (pessoa1.Equals(pessoa2)



Desta forma o objeto “pessoa2” é passado para o método Equals criado na classe Pessoa.



Se os objetos forem iguais retorna true.

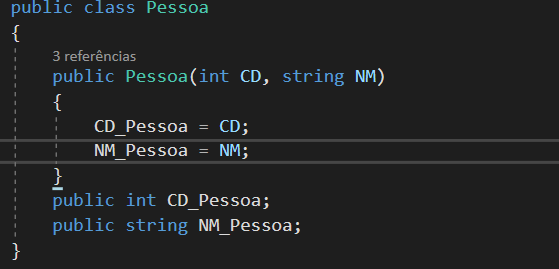


**Listas**

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 do tipo pessoa com um construtor e dois atributos, CD\_Pessoa e NM\_Pessoa.



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



Diferença entre List e IEnumerable

No tipo List implementa uma variedade de métodos capazes de adicionar, excluir, entre outros. Já o IEnumerable é do tipo readonly, ou seja, é do tipo somente leitura diferente do List que pode manipular seus dados.