# Herança 10/12/2020

É um tipo de associação que permite que uma classe herde dados e comportamentos de outra.

## Vantagens

* Reuso
* Polimorfismo

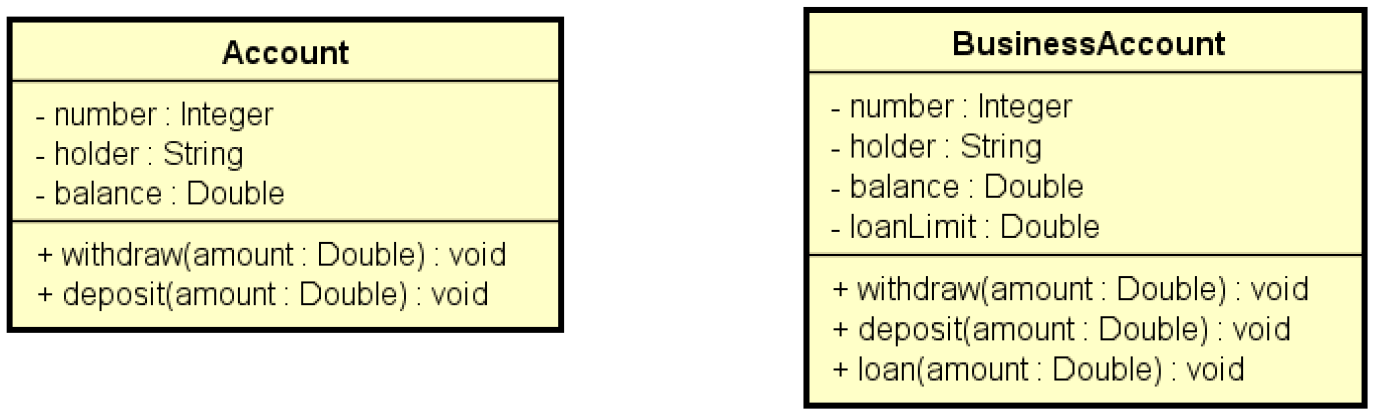
## Sintaxe

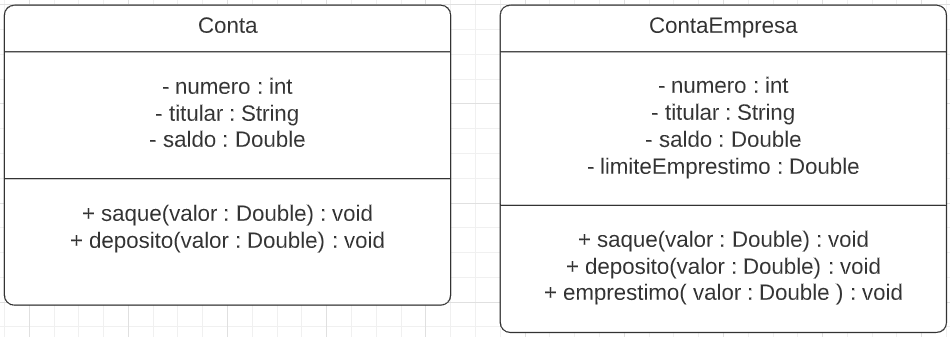
**:** Dois pontos, lemos **(estende)**

**base** A palavra base **(referência para a superclasse)**

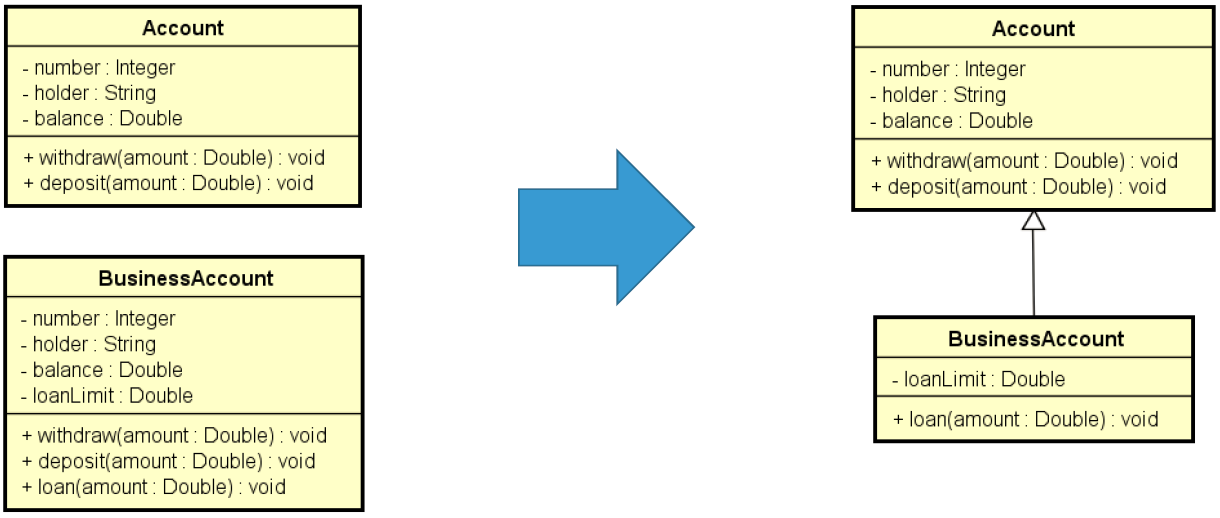
## Exemplo:

Suponha um negócio de banco que possui uma conta comum e uma conta para empresas, sendo que a conta para empresa possui todos membros da conta comum, mais um limite de empréstimo e uma operação de realizar empréstimo.





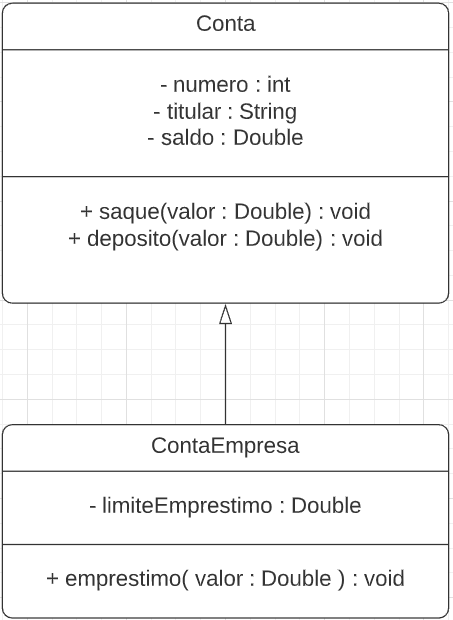
Herança permite o reuso de atributos e métodos (dados e comportamento). Ao invés de criar duas classes distintas repetindo atributos e métodos.



Conta Empresa herda da conta comum.

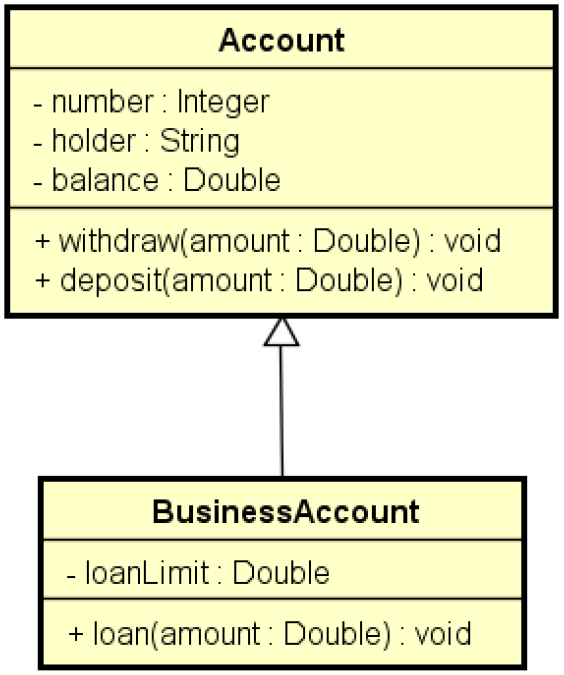
O símbolo é a seta com a ponta branca indicando que está herdando dela.

Dessa forma, todos os atributos e métodos da conta comum, também podem ser utilizados na conta Empresa. Mas não o contrário.



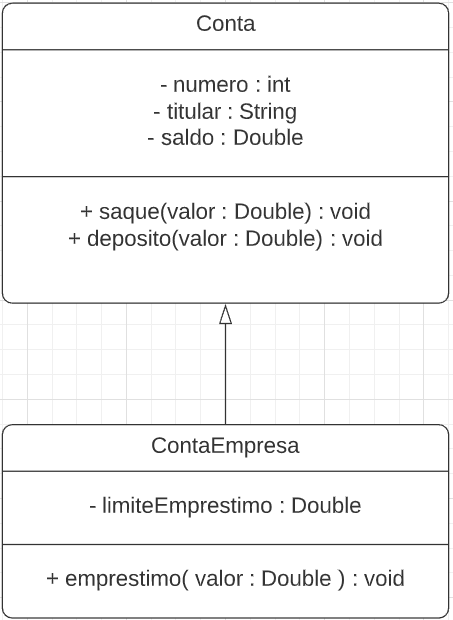
## Definições importantes:

* Relação "**é-um**"
  + Ou seja, a ContaEmpresa **é uma** Contasó que com algo mais.
* **Generalização/especialização**
  + Conta é uma **generalização**.
  + ContaEmpresa é uma **especialização**.
* Superclasse (classe base) / subclasse (classe derivada)
  + Conta é uma **superclasse**/classe base.
  + ContaEmpresaé uma **subclasse**/classe derivada.
* Herança / extensão
  + A contaEmpresa ela **estende** a conta comum. Porque ela possui tudo que a conta possui, mais algumas coisas.
* Herança é uma associação entre classes (e não entre objetos)

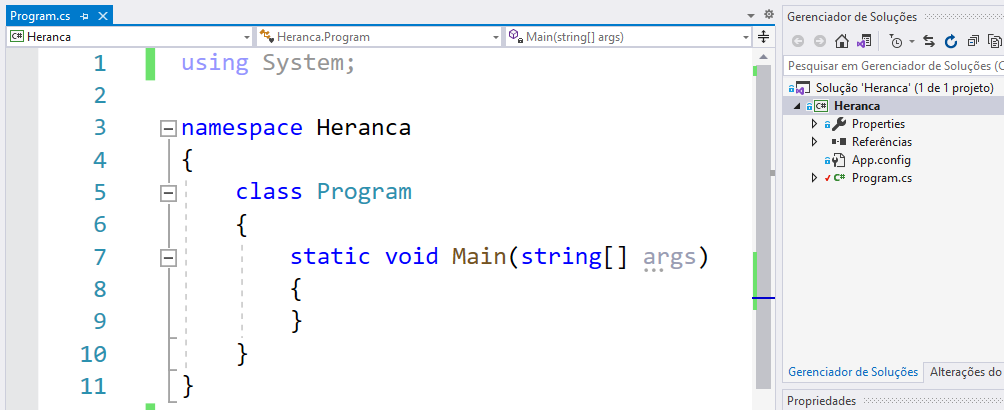


## Demonstração:

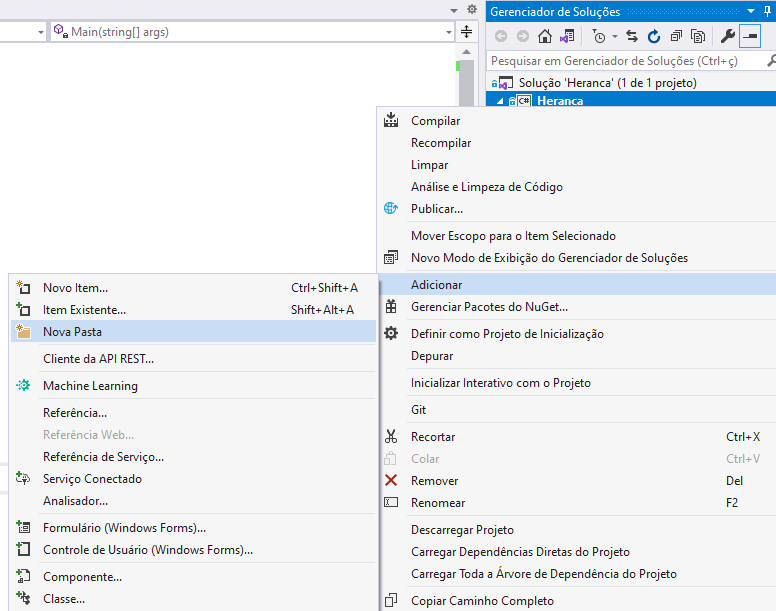
Implemente as classes Conta e ContaEmpresa e fazer alguns testes.



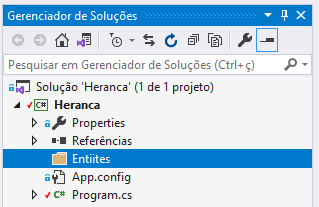
Vamos criar um projeto chamado **Heranca**



Botão direito no projeto. Adicionar New Folder.

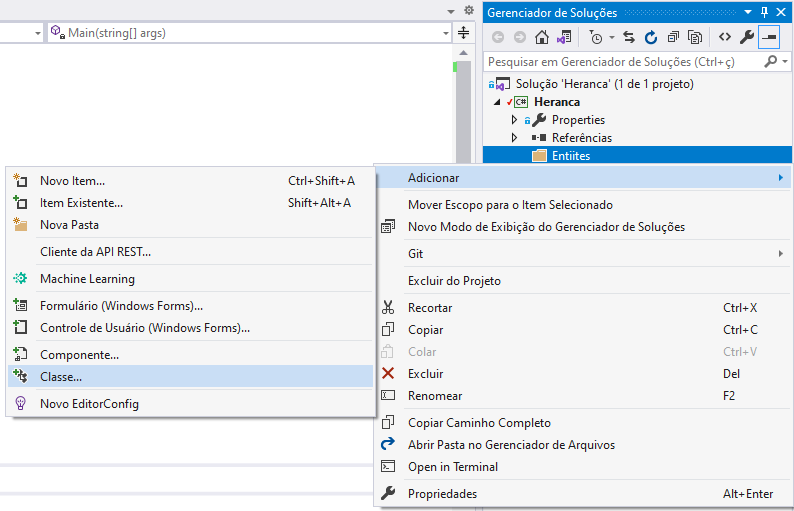


Crie a pasta Entities – que são as entidades.

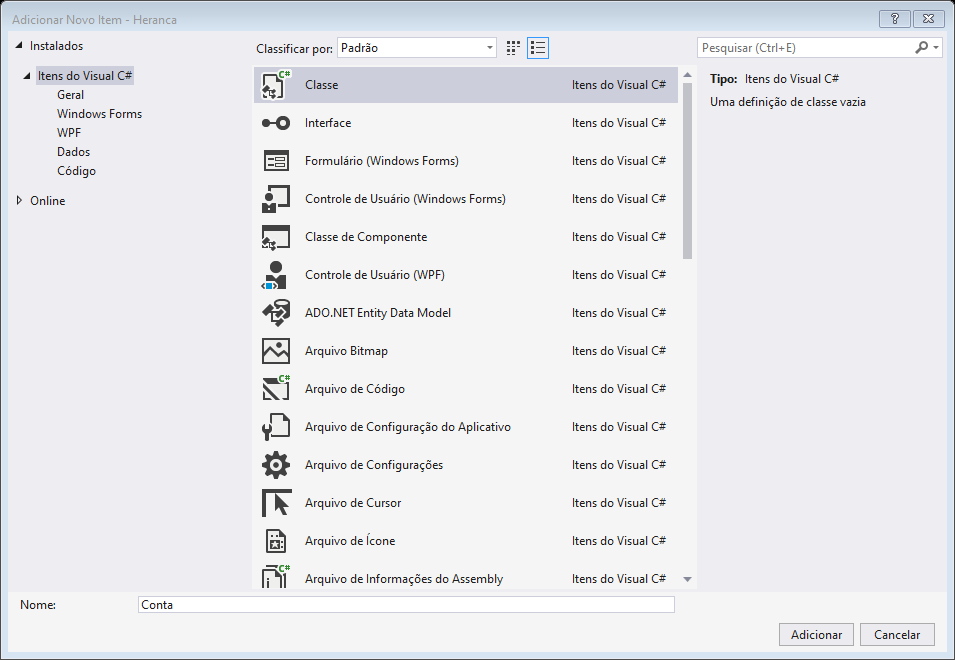


Para deixar nosso projeto melhor organizado, vamos separar as classes em uma pasta chamada Entities ou Entidades.

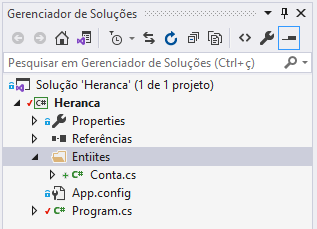
Botão direito no folder Entities Adicionar Classe



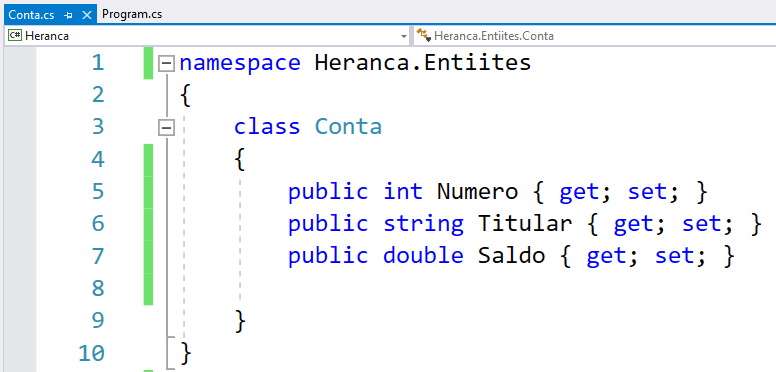
Adicione a classe Conta



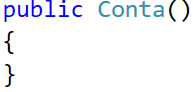
O projeto está assim:



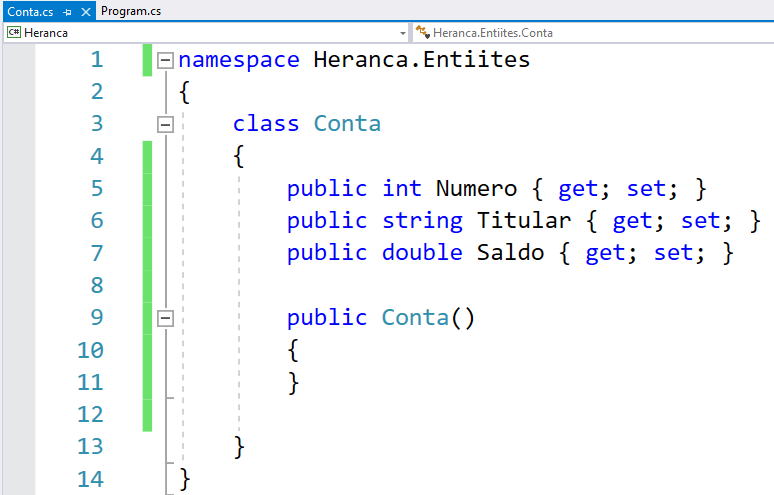
Criando as propriedades, os atributos:



Criando um construtor padrão

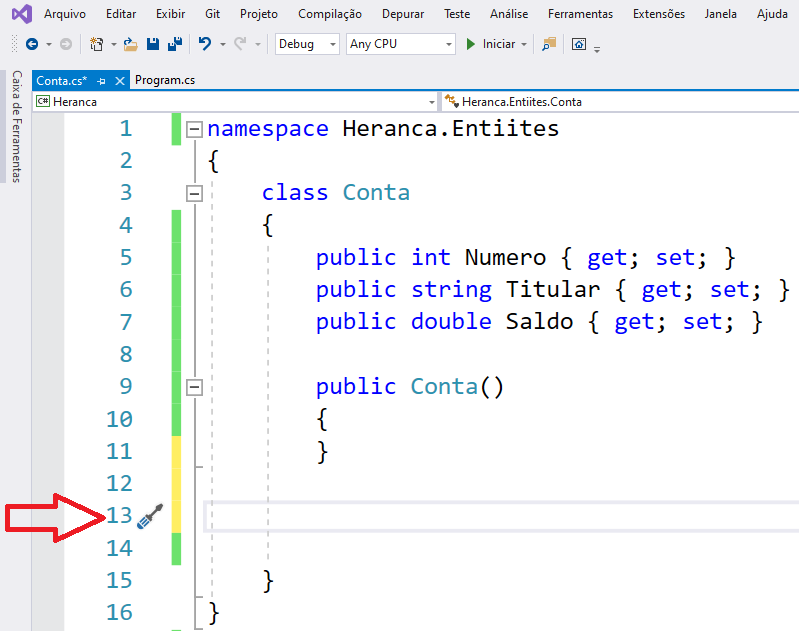


Fica assim:



Agora o construtor com os argumentos utilizando a ferramenta do Visual Studio.

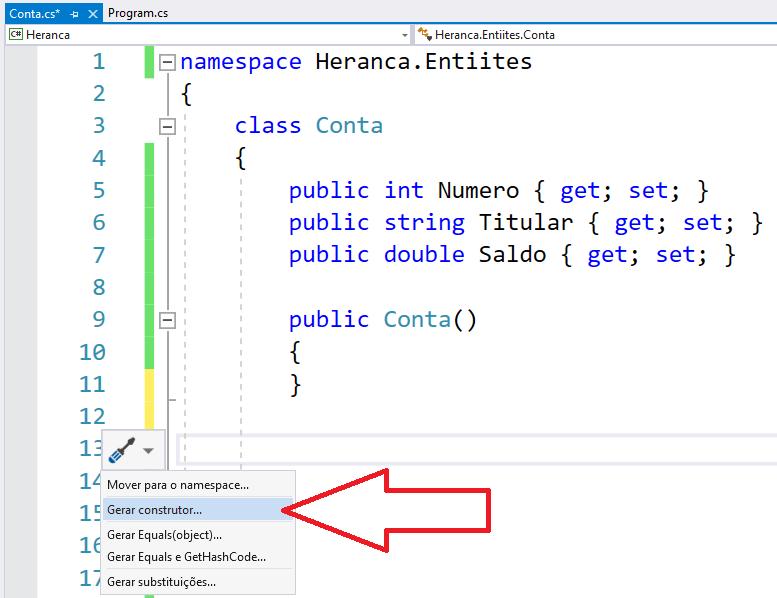
Coloque o cursor uma linha abaixo do construtor padrão. Irá aparecer o símbolo de uma chave de fenda. Clique nela.



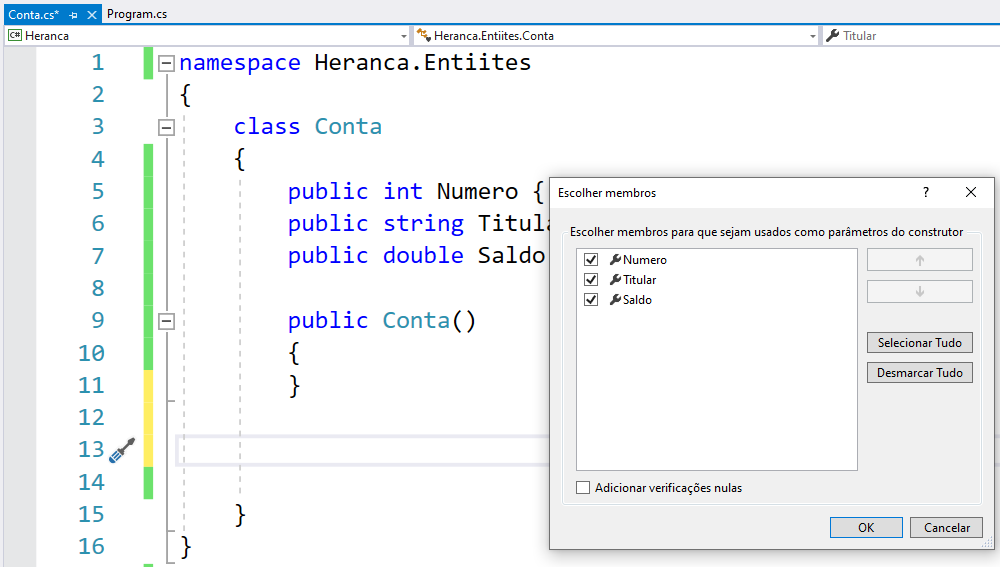
Ao passar o cursor na chave, irá aparecer a seta de um *drop down*. Clique nela.



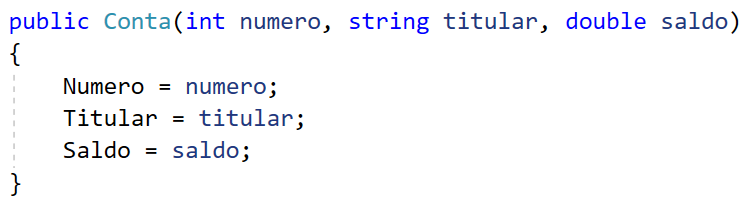
Escolha a opção Gerar construtor...



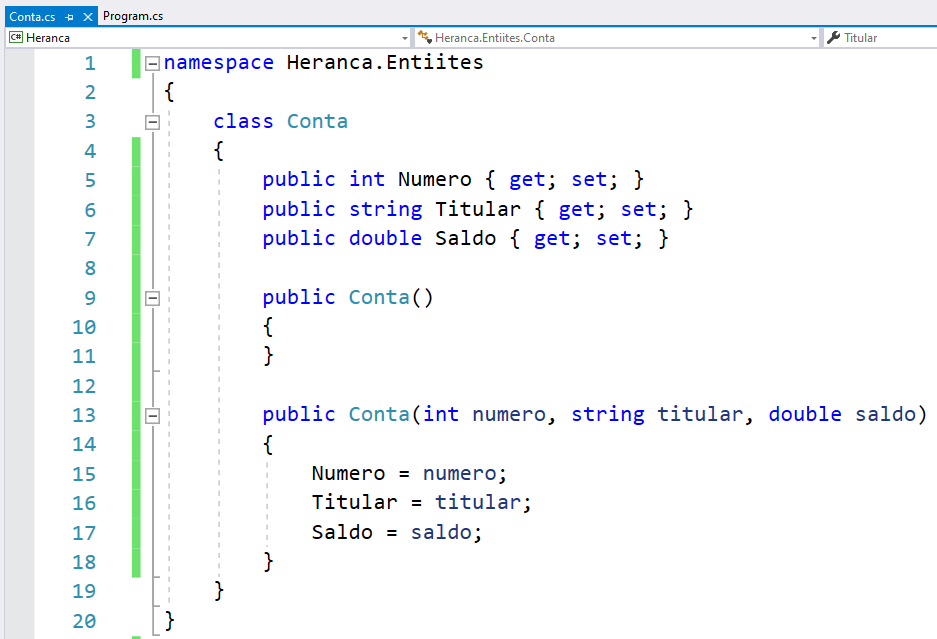
Pode deixar os três marcados. Clique em OK.



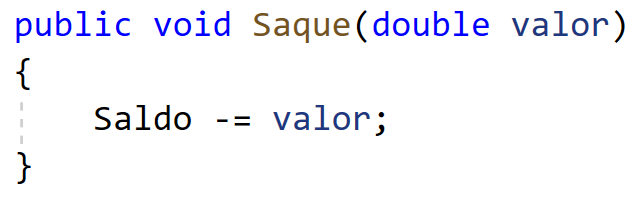
Os construtores foram criados automaticamente.



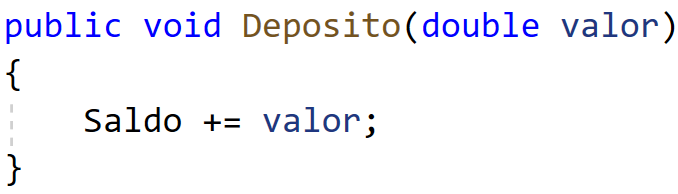
A classe Conta ficará assim:



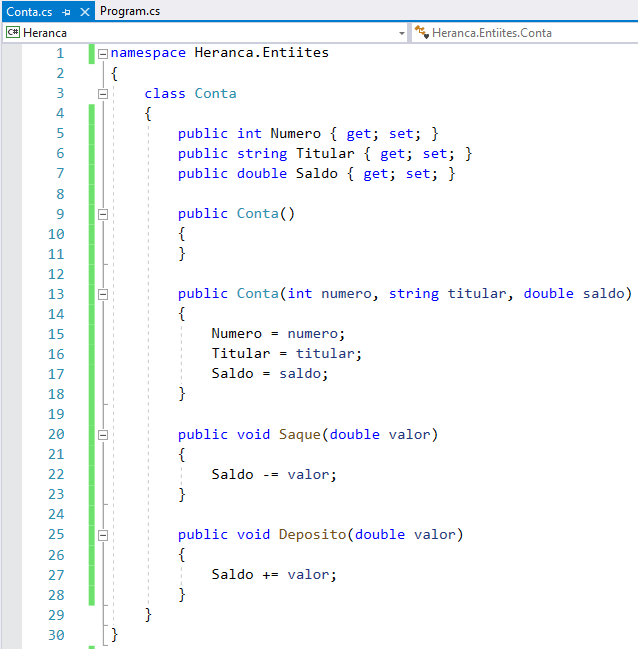
Vamos criar o método Saque, que é a retirada de um valor do saldo da conta.



Método Deposito, que é o fato de adicionar valor à conta.



A classe irá ficar assim:



## Código da classe Conta:

namespace Heranca.Entiites

{

class Conta

{

public int Numero { get; set; }

public string Titular { get; set; }

public double Saldo { get; set; }

public Conta()

{

}

public Conta(int numero, string titular, double saldo)

{

Numero = numero;

Titular = titular;

Saldo = saldo;

}

public void Saque(double valor)

{

Saldo -= valor;

}

public void Deposito(double valor)

{

Saldo += valor;

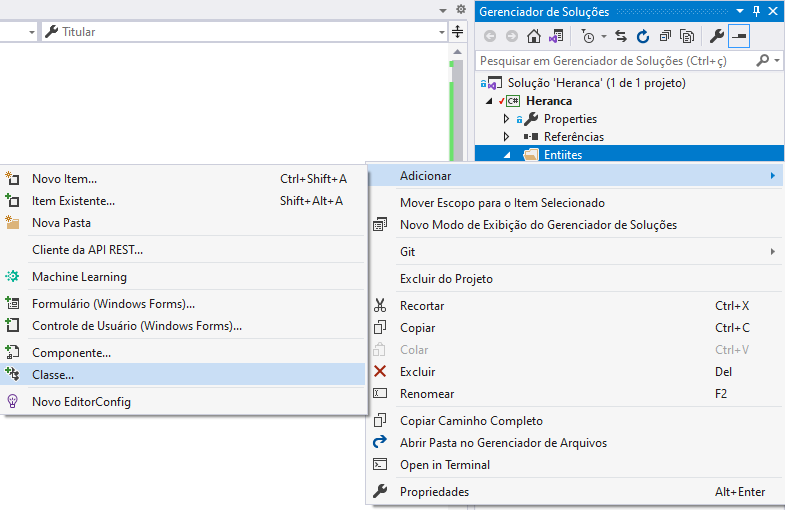
}

}

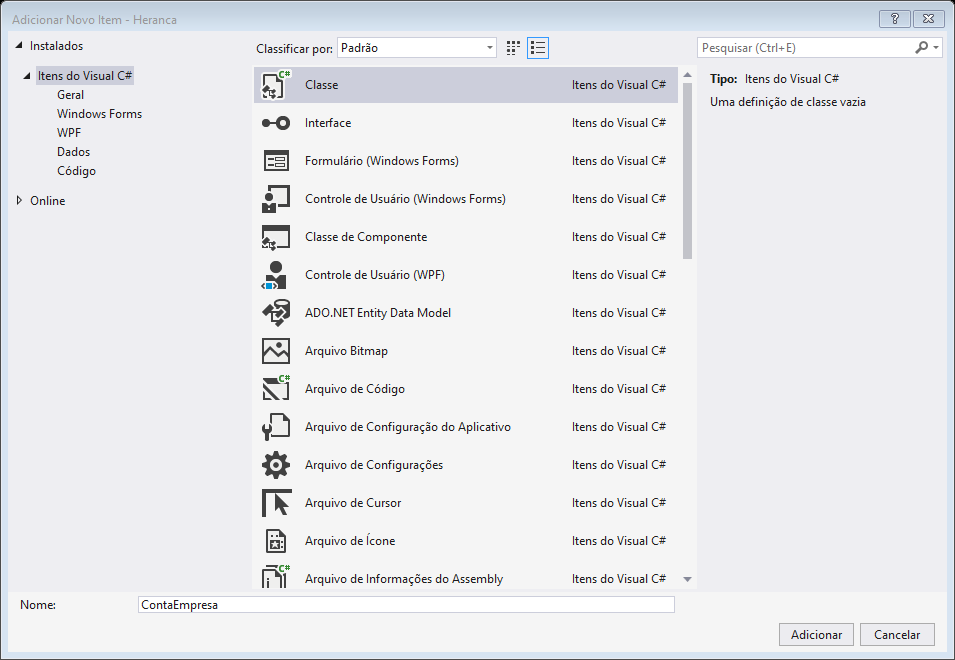
}

## Criando a classe ContaEmpresa

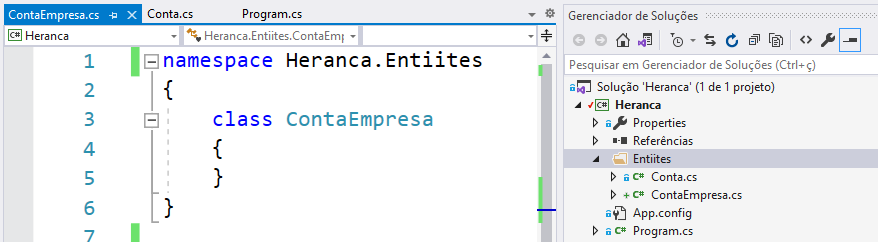
Botão direito na pasta Entities > Adicionar > Classe



Nomeando a classe como ContaEmpresa



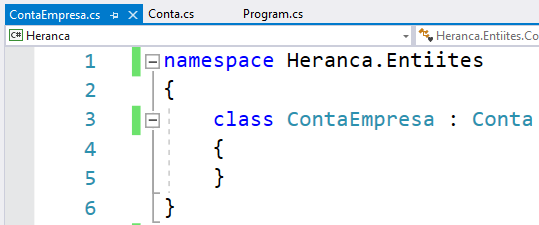
O projeto está ficando assim:



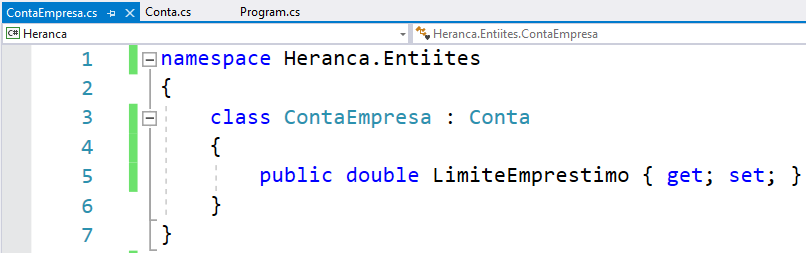
# : Expressando a Herança

Representando uma extensão da classe Conta. Criando a Herança.

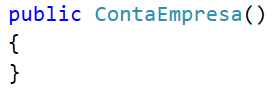
Basta incluir dois pontos ( **:** ) e o nome da superclasse na frente.



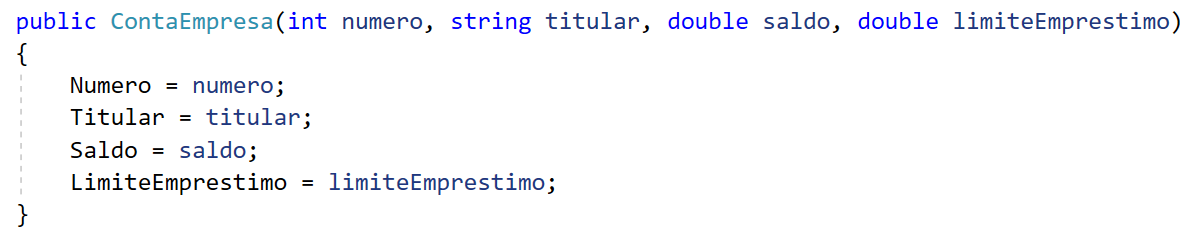
Criando o atributo LimiteEmprestimo



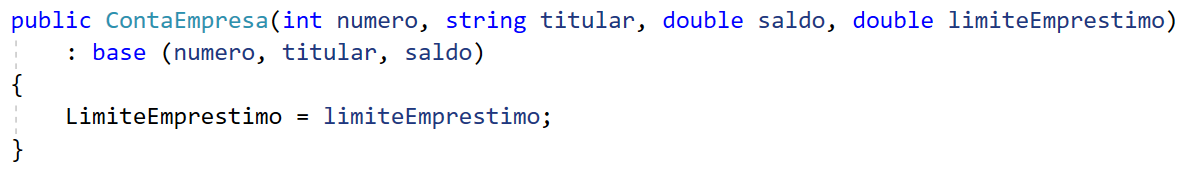
Construtor padrão. Para ter a opção de criar um objeto sem passar argumentos.



Uma opção **não** recomendada seria reescrever o construtor com todos os argumentos:

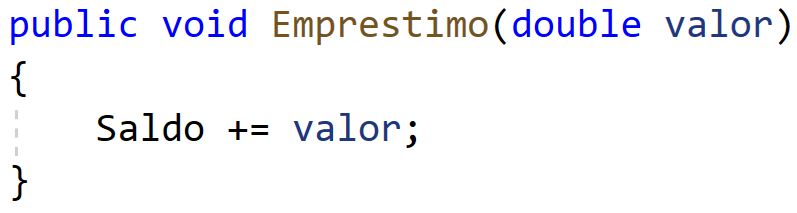


Porém o recomendado é criar o construtor passando apenas o argumento da classe e acionar os argumentos da superclasse com o **: base(argumentos)**

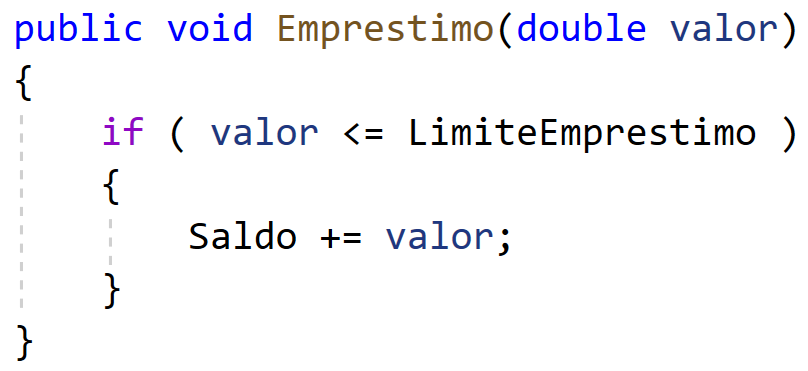


: base é para reutilizar os construtores da classe Conta sem reescrever.

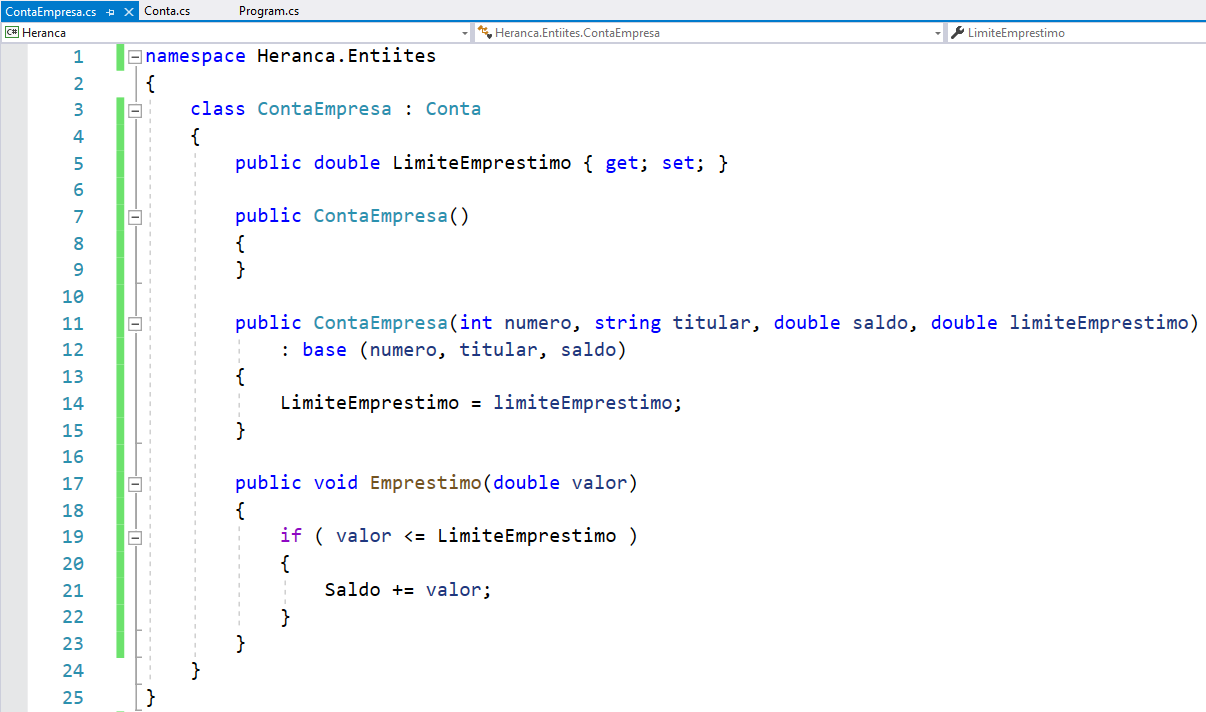
Agora vamos implementar o método Emprestimo. Acrescentar o valor do empréstimo no saldo da conta.



Porém, esse valor de empréstimo depende do limite de empréstimo. Portanto precisamos de uma condicional.



A classe ContaEmpresa ficará assim:



## Código da classe ContaEmpresa:

namespace Heranca.Entiites

{

class ContaEmpresa : Conta

{

public double LimiteEmprestimo { get; set; }

public ContaEmpresa()

{

}

public ContaEmpresa(int numero, string titular, double saldo, double limiteEmprestimo)

: base (numero, titular, saldo)

{

LimiteEmprestimo = limiteEmprestimo;

}

public void Emprestimo(double valor)

{

if ( valor <= LimiteEmprestimo )

{

Saldo += valor;

}

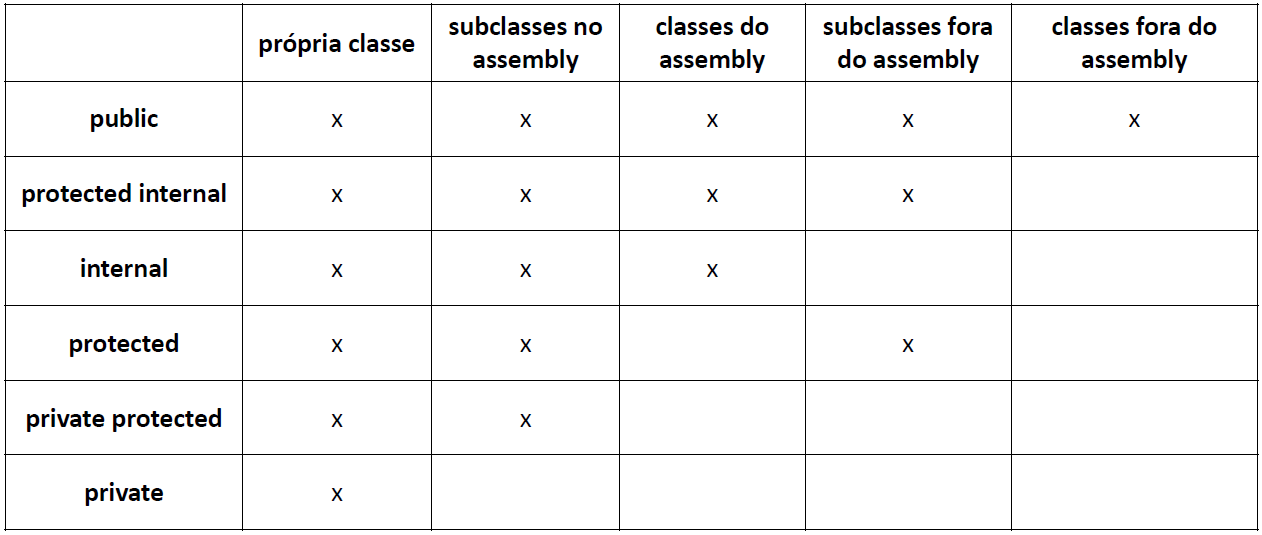
}

}

}

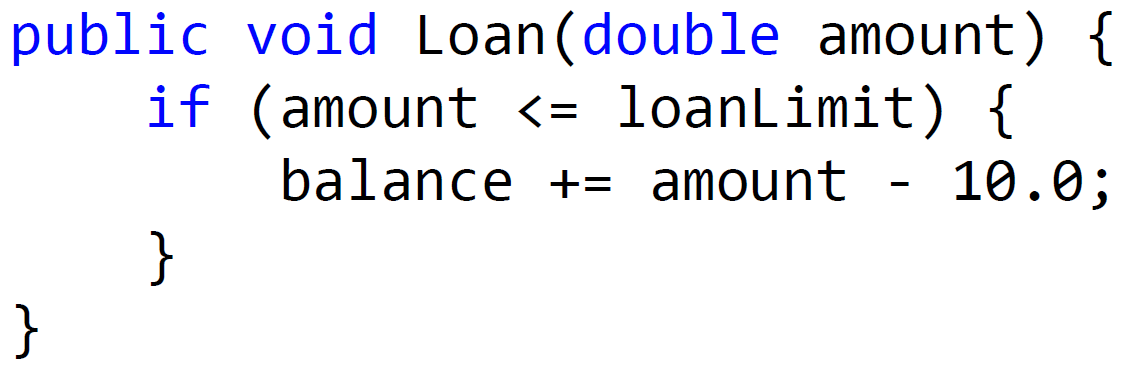
## Modificadores de acesso

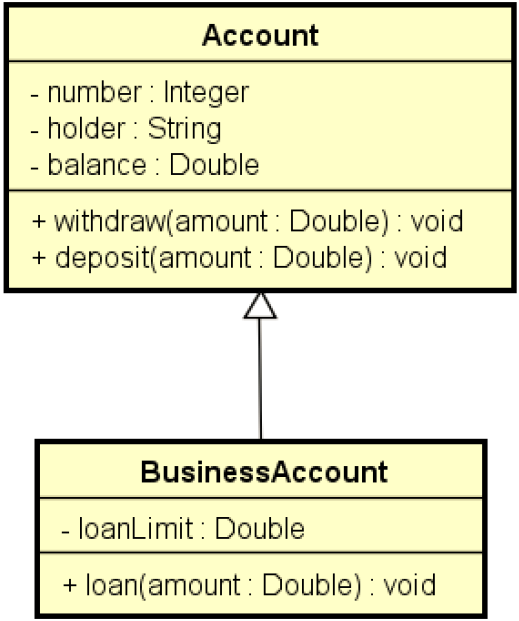
## protected



Suponha que, para realizar um empréstimo, é descontada uma taxa no valor de 10.0

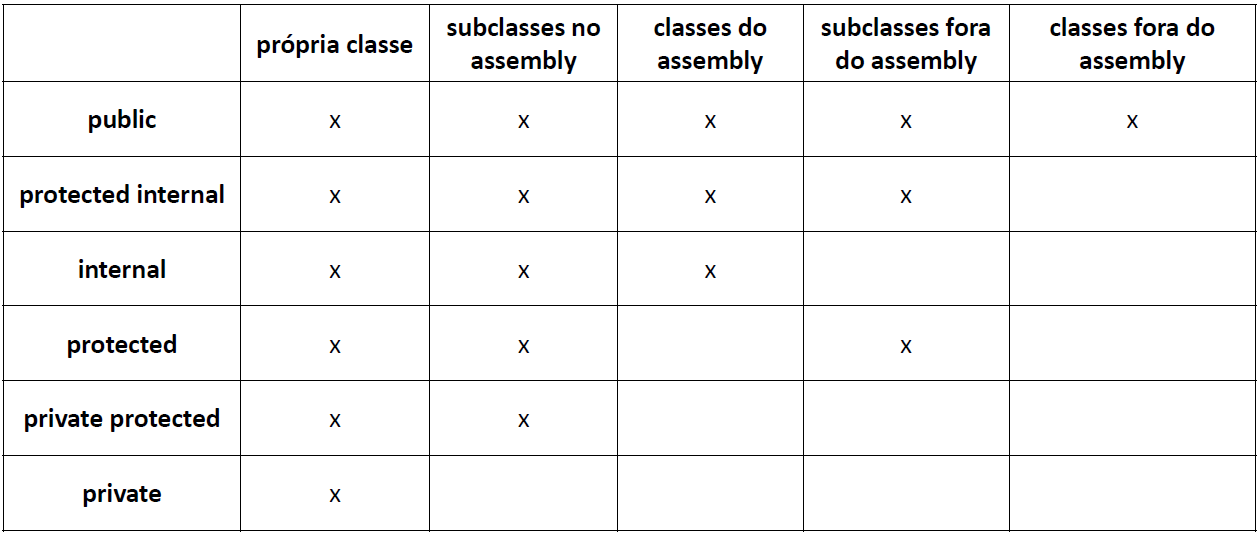
Isso resulta em erro:





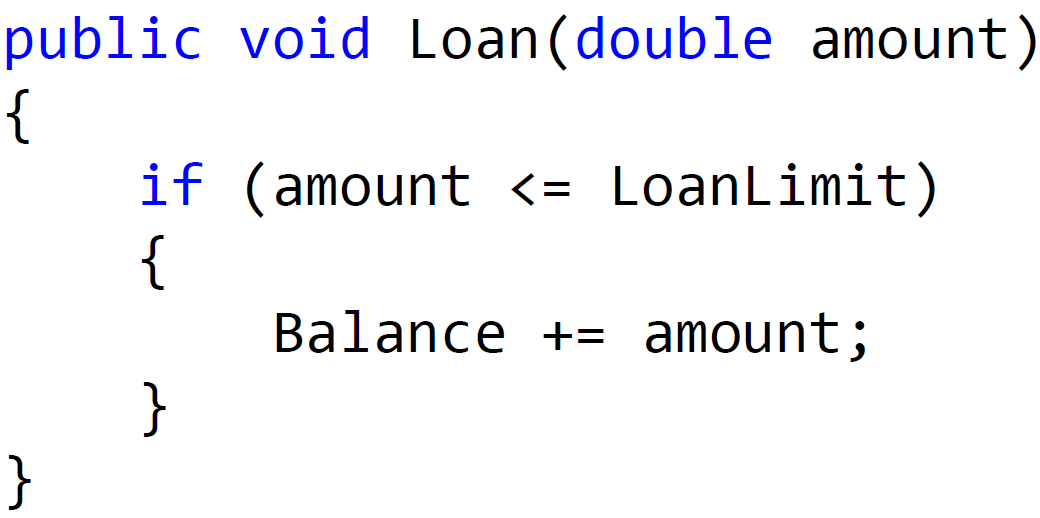
# Modificador de acesso protected

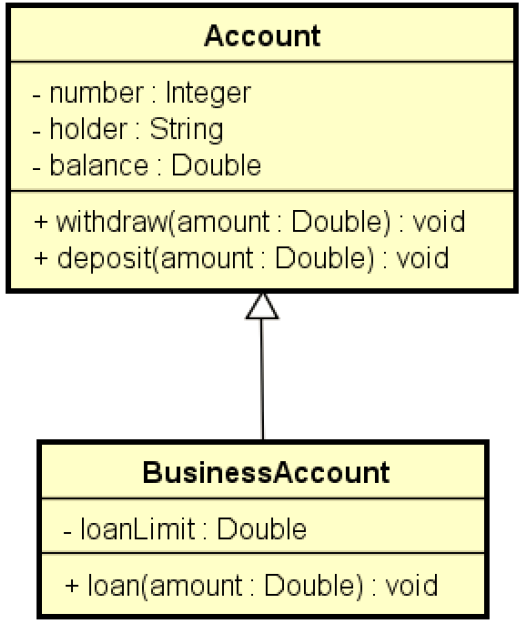
## Membros



## Exemplo:

Se o saldo tiver acesso privativo para alteração, isso resulta em erro:





# Referência:

Docs Microsoft – Matrizes multidimensionais (guia de programação C #) – <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/multidimensional-arrays > Acessado dia 03 de dezembro de 2020.