Programação Orientada a Objetos em C#

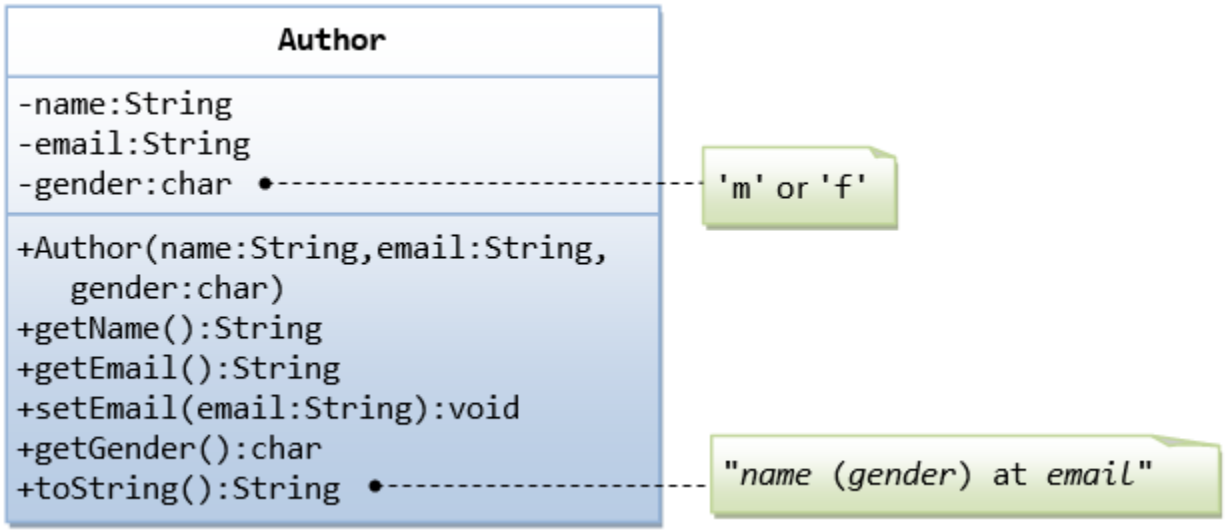
Teoria (Parte 01)

Aula de 05/07/2022

COTI Informática
Escola de Nerds

Classes

- As *classes* são os tipos do C# mais fundamentais. Uma classe é uma estrutura de dados que combina ações (métodos e outros membros da função) e estado (campos) em uma única unidade.
- Uma classe fornece uma definição para *instâncias* da classe criadas dinamicamente, também conhecidas como *objetos*.
- As classes dão suporte à *herança* e *polimorfismo*, mecanismos nos quais *classes derivadas* podem estender e especializar *classes base*.



Modificadores de Visibilidade

public

- Define acesso total para uma classe, atributo ou método.

internal

- Permite acesso somente dentro do mesmo Assembly.

protected

- Permite (para atributos ou métodos) acesso somente por meio de herança.

private

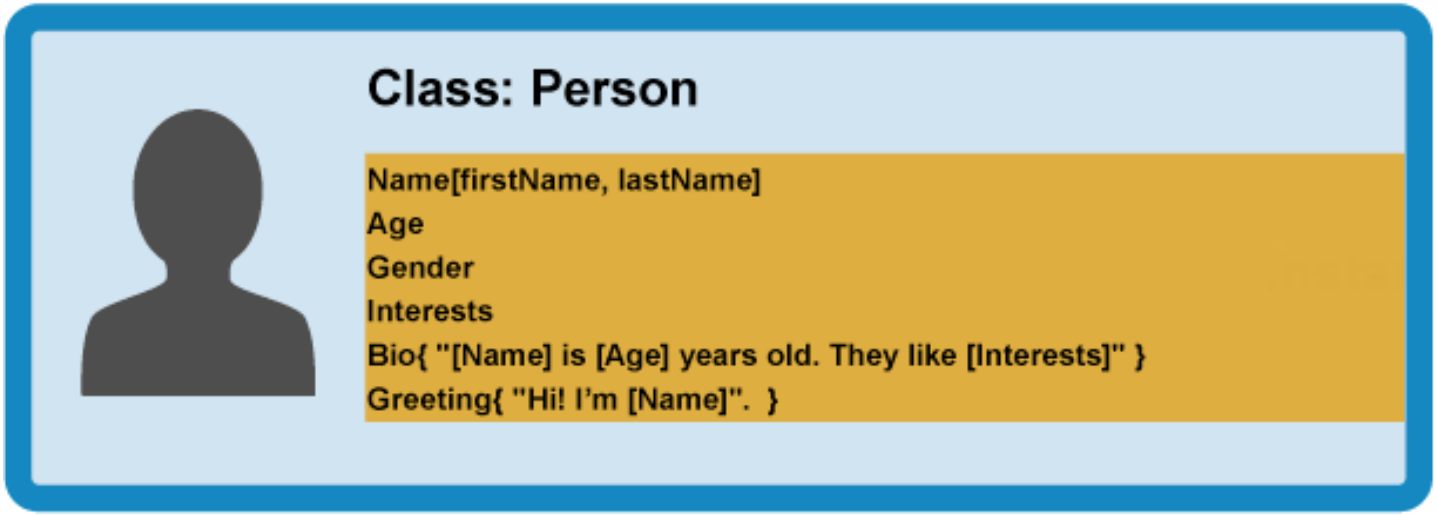
- Permite (para atributos ou métodos) acesso somente dentro da própria classe onde o elemento foi declarado.

Objeto

- Consiste de uma variável criada a partir do espaço de memória de uma classe. Também é chamado de instancia da classe.

```
var cliente = new Cliente();
```

[Objeto] [Criando espaço de memória - **Instância**]



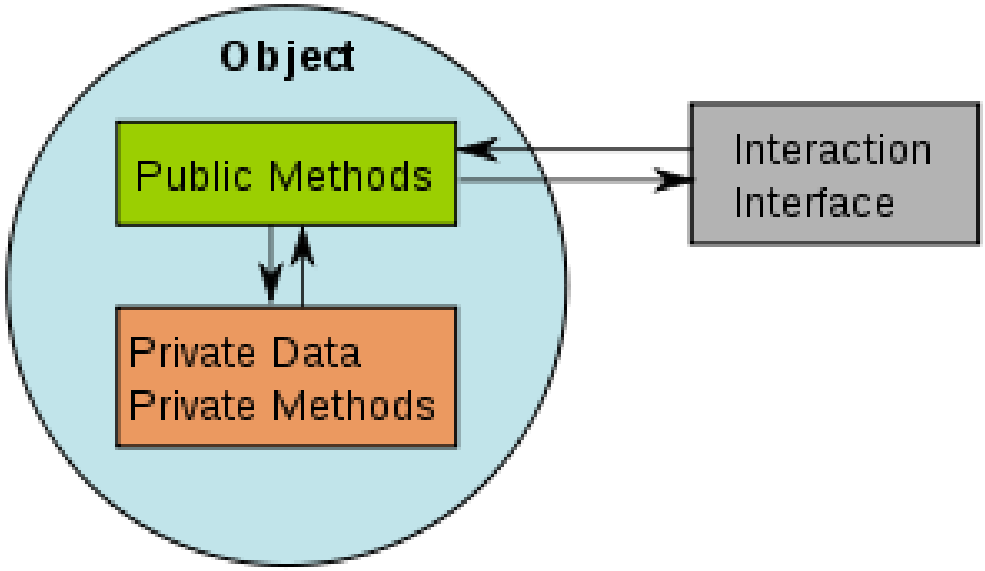
Encapsulamento

- Ao invés de declararmos os atributos como públicos, iremos mantê-los com visibilidade "private" e criar métodos que permitam acessar os atributos.

- Um exemplo de encapsulamento ocorre quando uma classe declara seus atributos como privados e cria métodos públicos que permitem acessar indiretamente os atributos.

- Estes métodos são chamados de **set** e **get**

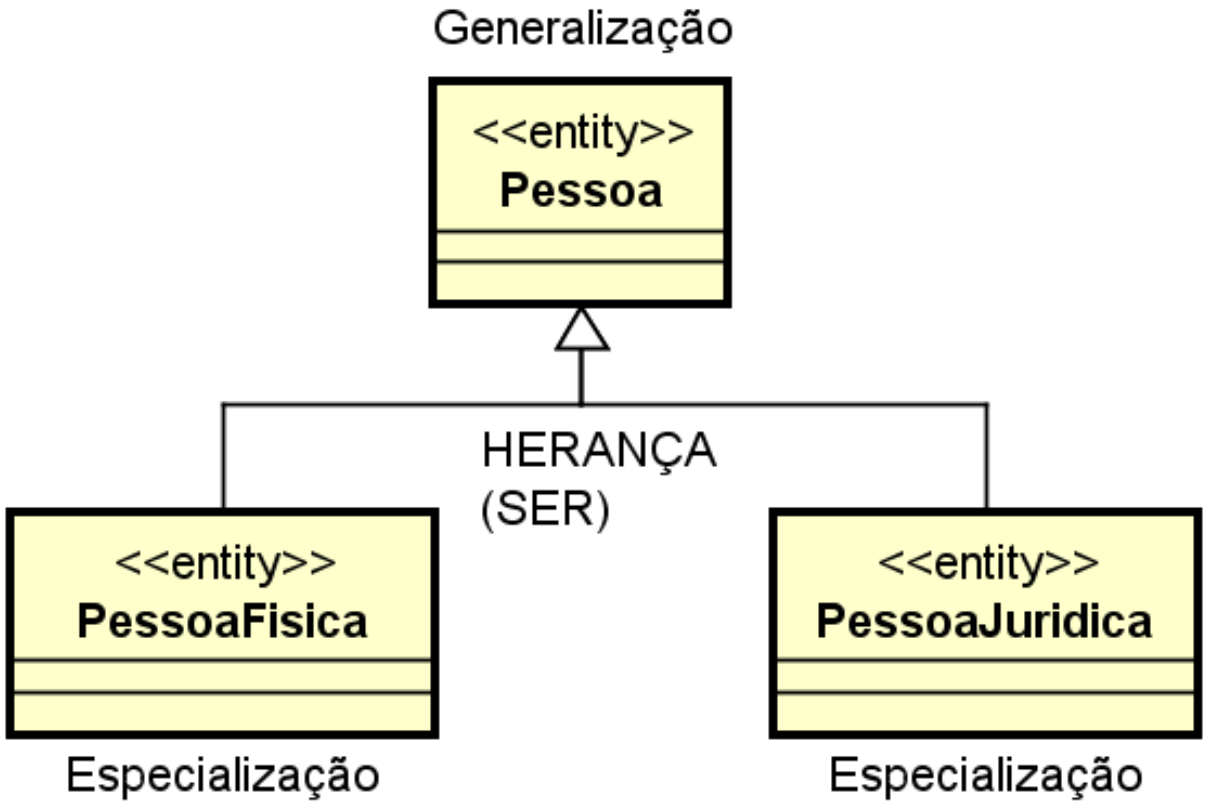
- **set** (entrada de dados)
- **get** (saida de dados)



Herança

- É um tipo de relacionamento entre classes que utiliza a abstração do verbo "SER". Define uma relação de hierarquia entre classes, também chamado de generalização / especialização.

Exemplo:

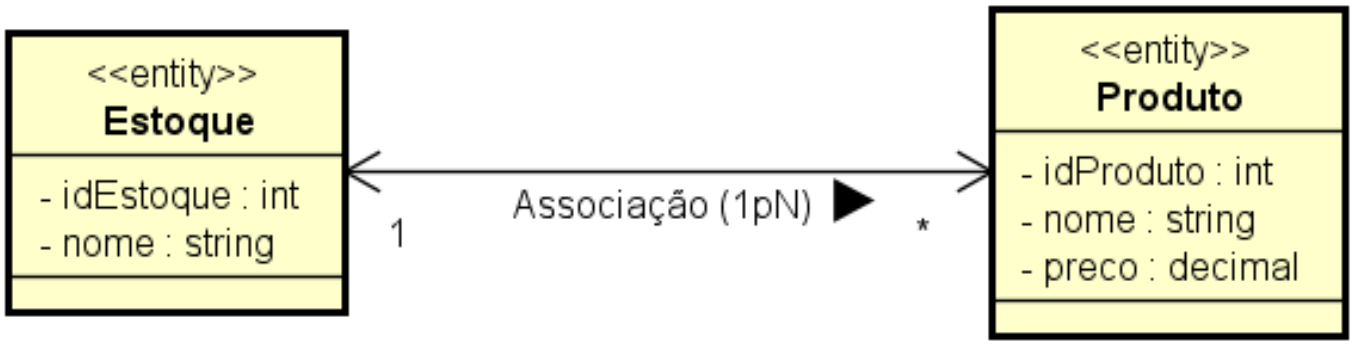


Relacionamento de Associação

Associação (TER)

É um relacionamento baseado na abstração do verbo TER, representa o conceito de utilização (TODO/PARTE) ao invés de herança.

Exemplo:



0..1	No mínimo zero e no máximo 1
1	1 e somente 1
0..*	No mínimo zero e no máximo muitos
1..*	No mínimo 1 e no máximo muitos
*****	Muitos

ENUMs

- Um tipo de enumeração (também chamado de uma enumeração ou enum) fornece uma maneira eficiente para definir um conjunto de constantes integrais nomeadas que podem ser atribuídas a um valor. Por exemplo, suponha que você precisa definir uma variável cujo valor representará um dia da semana. Há apenas sete valores significativos que essa variável armazenará. Para definir esses valores, você pode usar um tipo de enumeração, que é declarado usando a palavra-chave enum.

Exemplo:

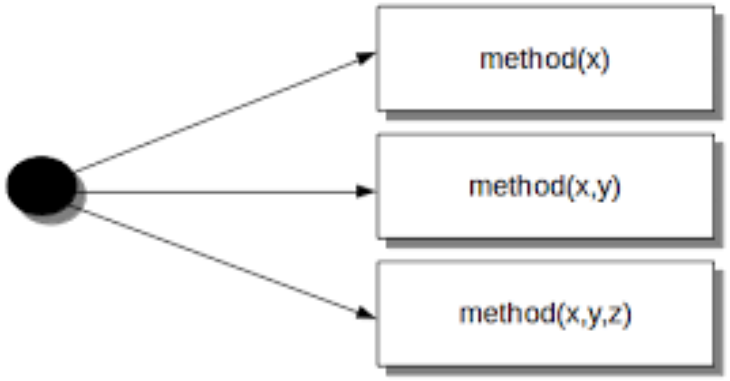
Sobrecarga de Métodos (**Overloading**)

- Recurso de programação orientada a objetos que permite ao desenvolvedor declarar métodos em uma classe com o mesmo nome, porem com entrada de argumentos diferentes.

  Por exemplo:

```
public class Calculo
{
   public double Somar(double a, double b)
   {
      return a + b;
   }

    public double Somar(double a, double b, double c)
    {
      return a + b + c;
   }
}
```



Sobrescrita de Métodos (**Override**)

- Recurso de programação orientada a objetos que permite ao desenvolvedor em uma classe "filha" (subclasse) reprogramar métodos herdados de sua "superclasse“

```
public class A
{
   public virtual void Imprimir()
   {
      Console.WriteLine("Imprime A");
   }
}

public class B : A
{
   public override void Imprimir()
   {
      Console.WriteLine("Imprime B");
   }
}
```

`virtual`

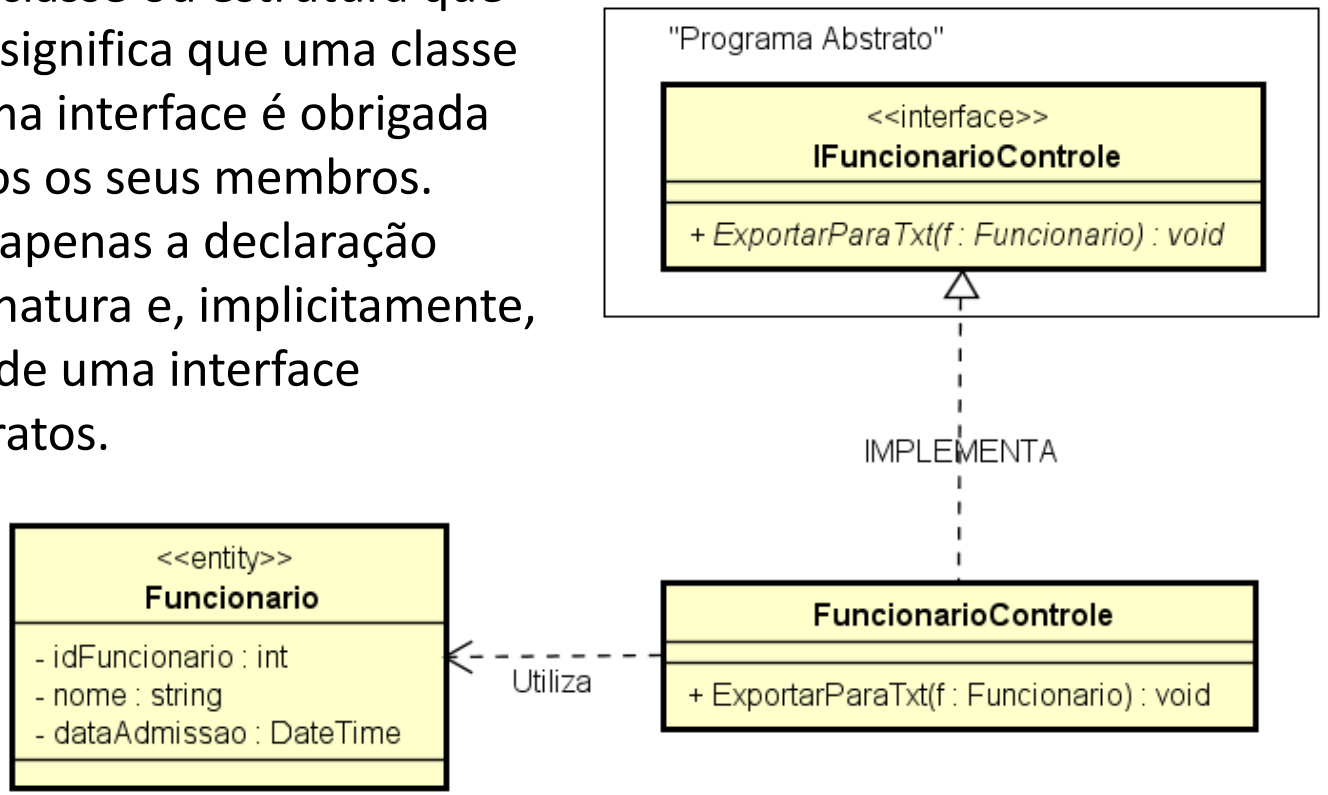
É uma palavra reservada da linguagem utilizada para permitir que um método seja sobrescrito por uma subclasse.

Para que haja sobrescrita de método, a superclasse sempre deverá declarar o método que seja permitir sobrescrever como "virtual"

Interfaces

Interface é uma ferramenta de programação orientada a objetos utilizado para definir padrões (contratos) que as classes deverão implementar.

Uma interface funciona como um contrato entre si e qualquer classe ou estrutura que a implementa. Isso significa que uma classe que implementa uma interface é obrigada a implementar todos os seus membros. Uma Interface tem apenas a declaração de membro ou assinatura e, implicitamente, todos os membros de uma interface são públicos e abstratos.

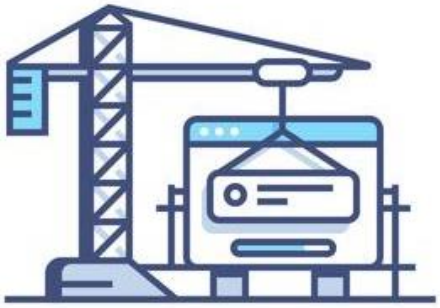


Regras sobre interfaces:

- Interfaces não podem ter atributos
- Interfaces não podem ter construtores
- Métodos de interface devem ser abstratos, ou seja, não podem ter corpo, apenas assinatura.
- Métodos de interface já são implicitamente públicos.

Regra principal:

Quando uma classe HERDA uma interface,
a classe é obrigada a implementar (fornecer corpo)
para todos os métodos abstratos da interface.



Tratamento de Exceções

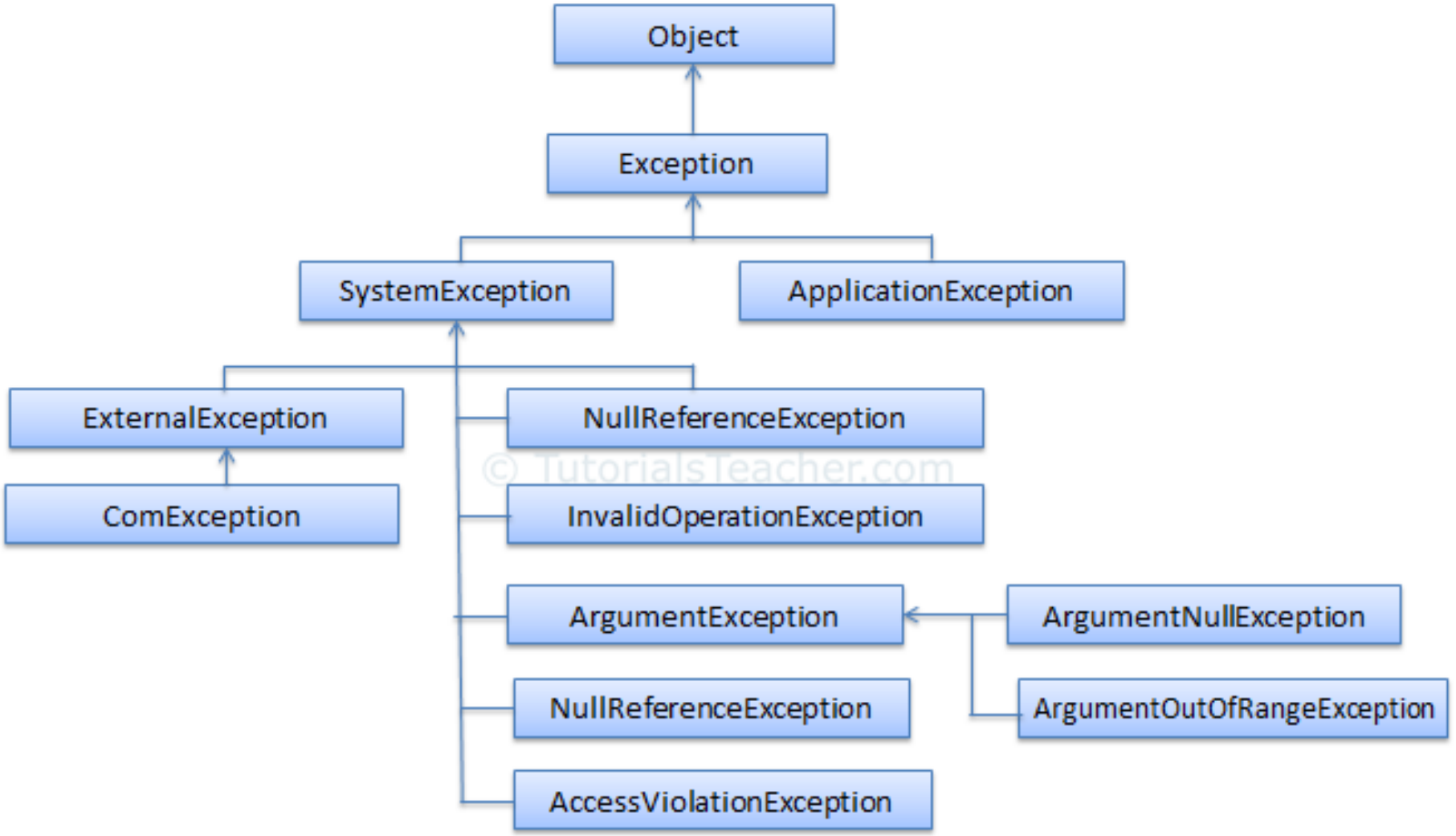
São erros que podem ocorrer em um programa não durante a sua compilação mas sim durante a sua execução.

Estes erros em tempo de execução são chamados de **Exceções**. Para tratarmos estas exceções podemos utilizar um bloco de programação denominado **try / catch**

Exception

Classe que representa qualquer tipo
de erro ocorrido em tempo de execução
(qualquer tipo de exceção.)

Hierarquia de Exceções



1. ENTENDENDO O DIAGRAMA DE CLASSES.

Em UML, o **diagrama de classes** é uma representação da estrutura e relações das classes que servem de modelo para objetos. É o principal diagrama para representar uma modelagem dentro do paradigma orientado a objetos. É uma modelagem muito útil para o desenvolvimento de sistemas, pois define todas as classes que o sistema necessita possuir e é a base para a construção dos diagramas de comunicação, sequência e estados por exemplo. A **Classe** é uma estrutura que representa um conjunto de objetos. A classe contém a especificação do objeto; suas características: atributos e métodos (ações / comportamentos). Por exemplo:

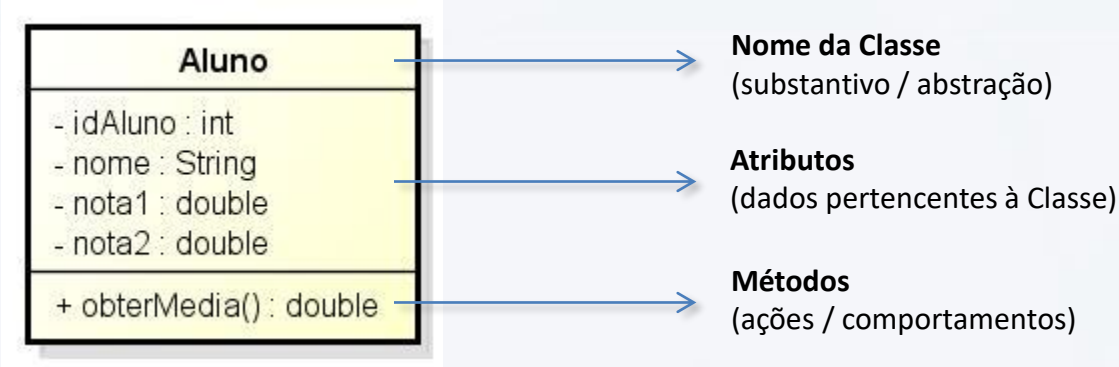
Modificadores de visibilidade:
Definem o nível de permissão de acesso
Aos elementos da Classe. São eles:

- **private**
Acesso permitido somente dentro da própria Classe (Nível mais restritivo)

~ **default / package**
Acesso permitido somente dentro do pacote onde a Classe foi criada

**protected**
Acesso permitido dentro do pacote ou por meio de herança.

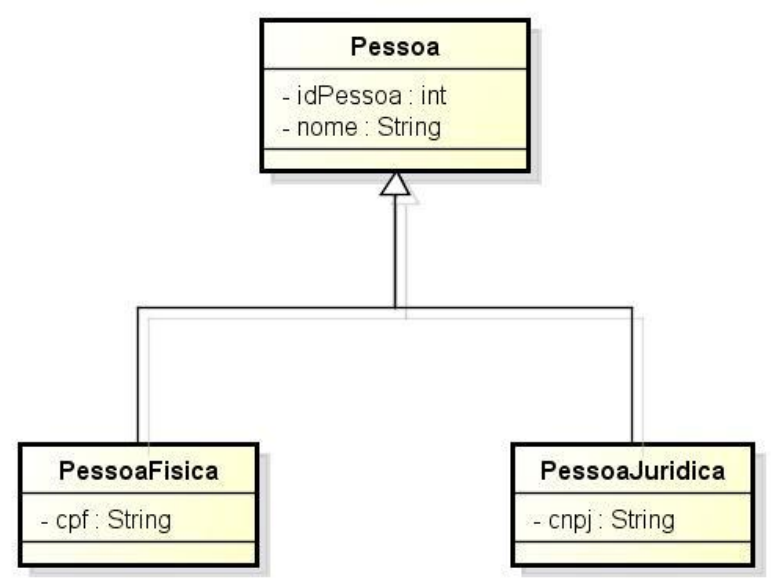
+ **public**
Acesso permitido em qualquer nível.



2. RELACIONAMENTOS ENTRE CLASSES.

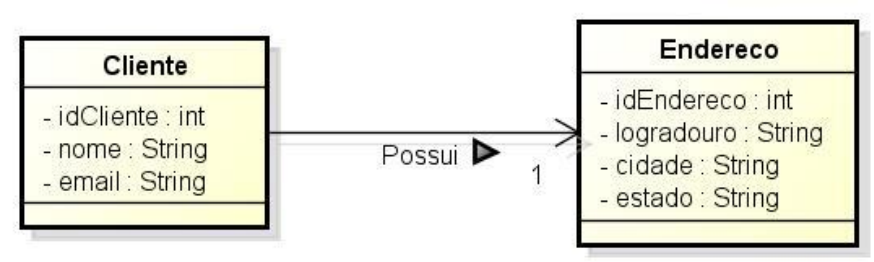
Podemos dizer que classes podem relacionar-se de 2 maneiras: **SER** ou **TER**, sendo o primeiro chamado de **Herança** (Generalização / Especialização) e o segundo de **Associação** (Todo / Parte). Por exemplo:

Herança
(Generalização / Especialização)



Podemos afirmar que **PessoaFisica** e **PessoaJuridica** "herdam" Pessoa, ou seja, são especializações da superclasse Pessoa

Associação
(Todo/ Parte)



Podemos afirmar que **Cliente** "possui" **Endereco**, ou seja, a Classe Endereço faz parte da Classe Cliente.
Note que podemos definir a **multiplicidade** deste tipo de relacionamento, são eles:

0..1	No mínimo zero e no máximo 1
1	1 e somente 1
0..*	No mínimo zero e no máximo muitos
1..*	No mínimo 1 e no máximo muitos
*****	Muitos