Agenda

- ❏ Encapsulamento
- ❏ Modificadores de Acesso
- ❏ Herança

Encapsulamento (Encapsulation, Data Hiding)

→ É aplicado aos atributos e métodos de uma classe.

→ Consiste em proteger os dados, ou até mesmo escondê-los.

→ Para limitar, ou controlar o conteúdo de um atributo, métodos devem ser utilizados para atribuir ou alterar valores dos atributos de um objeto.

→ O uso de atributos diretamente pelos clientes de uma classe é desencorajado.

→ Dependendo da visibilidade, o acesso aos atributos não pode ser feito diretamente, mas indiretamente.

Encapsulamento (benefícios)

❏ Esconde os detalhes da implementação de uma classe.

❏ Força o usuário a usar um método para acesso aos dados.

❏ Permite definir o modo de acesso aos dados

→ leitura

→ escrita

→ leitura/escrita

❏ Proteger os dados que estão dentro dos objetos, evitando assim que os mesmos sejam alterados erroneamente

Encapsulamento

O uso de métodos de leitura (get) e escrita (set) visam desacoplar os atributos de uma classe dos clientes que a utilizam, tornando-os assim uma propriedade.

Nos exemplos exibidos a seguir o atributo idade está encapsulado:

```
0 referências
public class Pessoa
{
    private int idade;

    0 referências
    public int Idade
    {
        get
        {
            return idade;
        }
        set
        {
            idade = value;
        }
    }
}
```

OU

```
0 referências
public class Pessoa
{
    0 referências
    public int Idade { get; set; }

}
```

Modificadores de Acesso (Visibilidade)

As linguagens OO disponibilizam formas de controlar o acesso aos membros - atributos e métodos - de uma classe. No mínimo, devemos poder fazer diferença entre o que é público e o que é privado.

O C# disponibiliza três modificadores de acesso:

- ➢ private
- ➢ protected
- ➢ public

Quando nenhum modificador é utilizado dizemos que o membro está com o nível de acesso **private**

Modificadores de Acesso (Visibilidade)

Membros públicos podem ser acessados indiscriminadamente, enquanto os privados só podem ser acessados pela própria classe.

Por hora vamos focalizar nossa atenção em private e public, pois o nível de acesso protected será abordado na próxima aula.

Modificadores de Acesso

Símbolo	Palavra-chave	Descrição
-	**private**	Atributos e métodos são acessíveis somente nos métodos da própria classe. Este é o nível mais rígido de encapsulamento.
#	**protected**	Atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses.
+	**public**	Atributos e métodos são acessíveis em todos os métodos de todas as classes. Este é o nível menos rígido de encapsulamento.

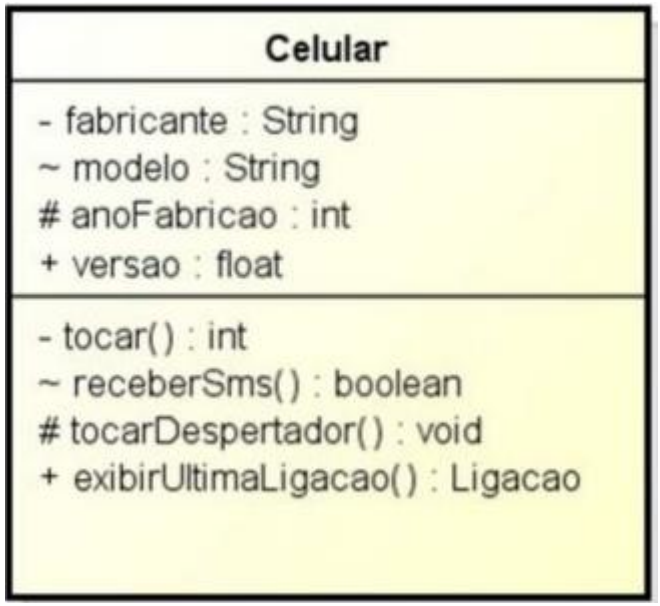
Modificadores de Acesso (Visibilidade)

Os modificadores de acesso podem ser representados no diagrama de classes através dos símbolos:

- (private)

 # (protected)

+ (public)



Modificadores de Acesso (Visibilidade)

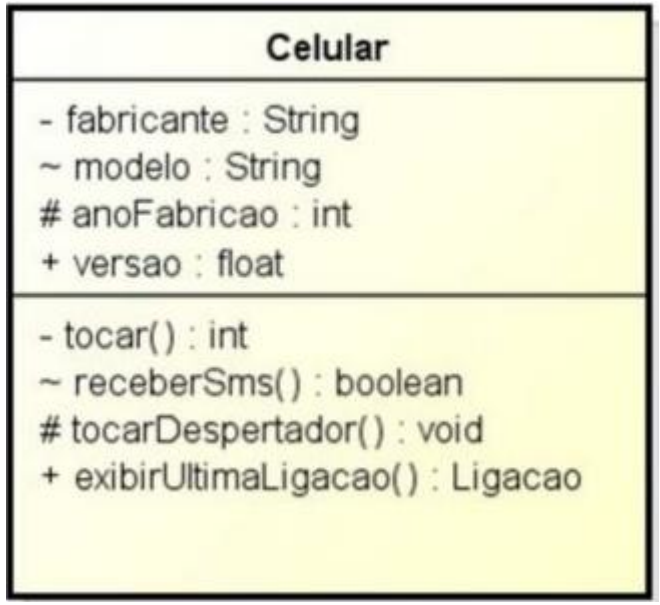
```
0 referências
public class Celular
{
    0 referências
    private string Fabricante { get; set; }
    0 referências
    string Modelo { get; set; }
    0 referências
    protected int AnoFabricacao { get; set; }
    0 referências
    public float Versao { get; set; }
    0 referências
    private int Tocar()
    {

    }
    0 referências
    bool ReceberSms()
    {

    }
    0 referências
    protected void TocarDespertador()
    {

    }
    0 referências
    public Ligacao ExibirUltimaLigacao()
    {

    }
}
```



Herança

Herança é um dos mecanismos fundamentais para as linguagens que suportam o paradigma OO.

Este mecanismo possibilita a criação de novas classes a partir de uma já existente.

A herança é utilizada como forma de reutilizar os atributos e métodos de classes já definidas, permitindo assim derivar uma nova classe mais especializada a partir de outra classe mais genérica existente.

Aplicar herança sempre envolve basicamente dois elementos: uma superclasse (classe pai) e uma subclasse (classe filha).

Herança

Superclasse é também conhecida como classe ancestral ou classe pai. Apresenta as características genéricas de um conjunto de objetos.

Subclasse é também conhecida como classe descendente ou classe filha. Ela estende a superclasse para incluir suas características.

A **subclasse**:

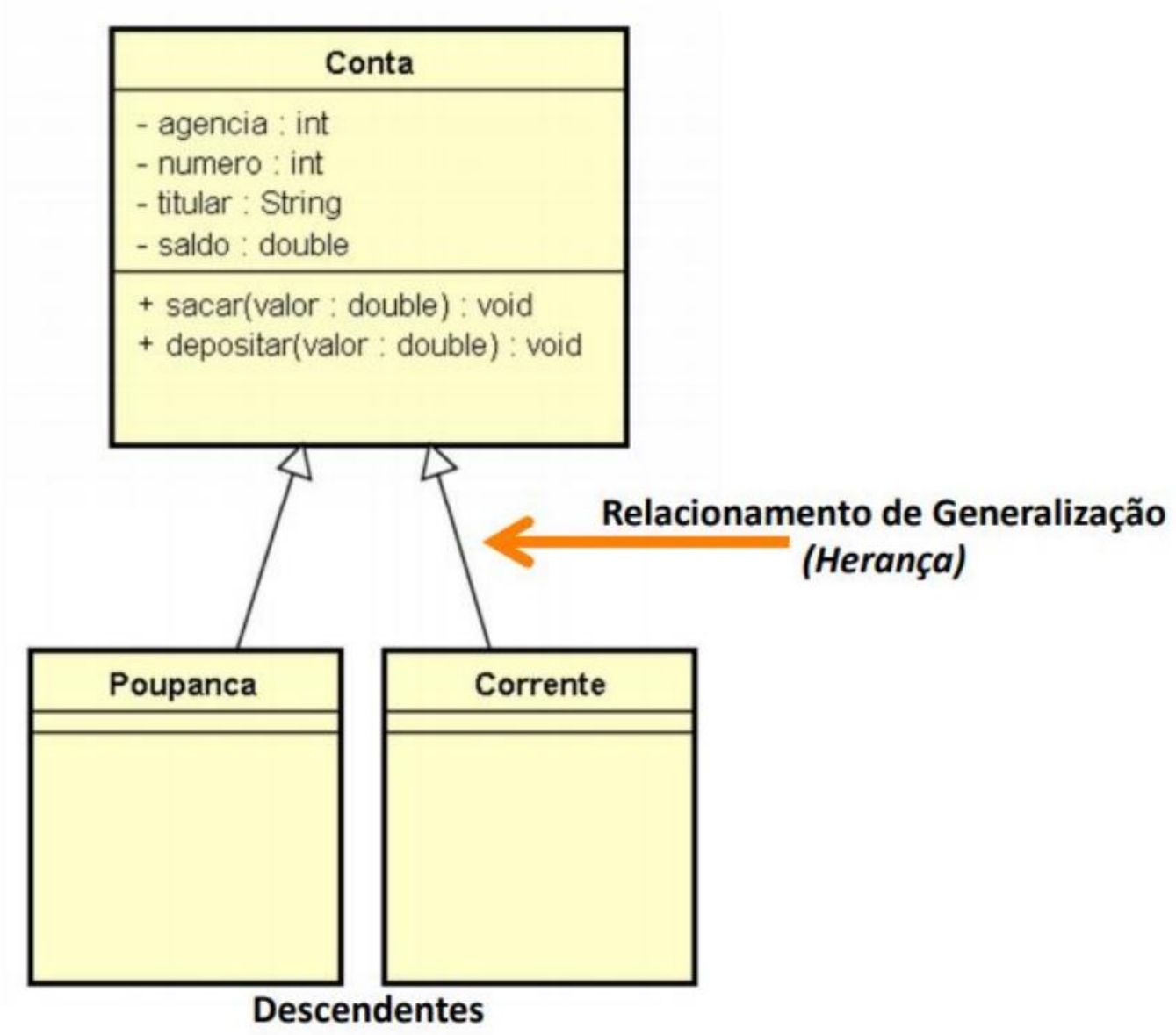
– Herda os atributos (desde que não sejam privados)

– Permite adicionar novos atributos (que será visível somente na subclasse)

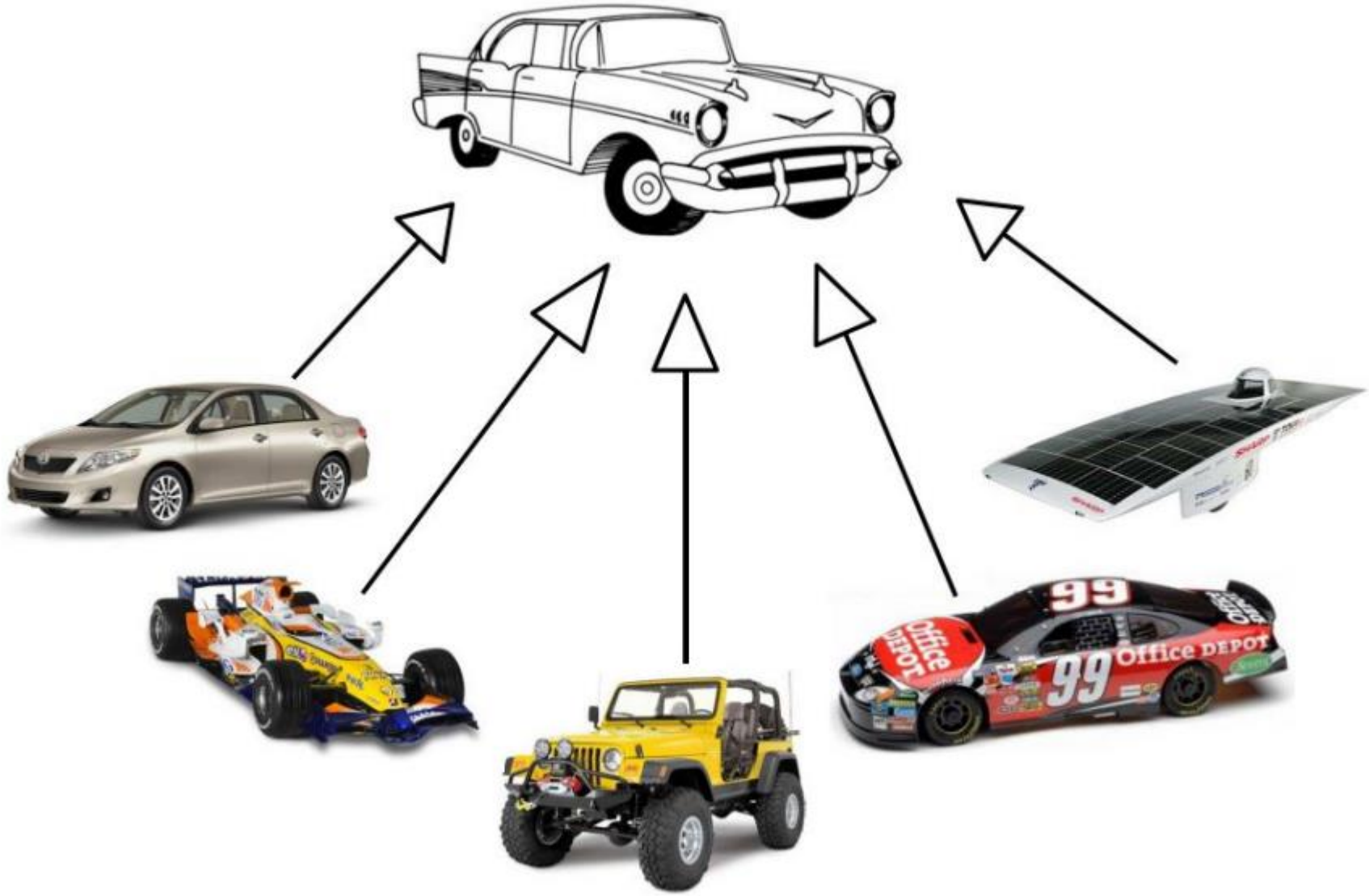
– Em relação aos métodos, a subclasse poderá utilizá-los/herdá-los (superclasse), bem como criar novos métodos e alterá-los

– Métodos construtores não são herdados (porém podemos chamá-los dentro do construtor da subclasse)

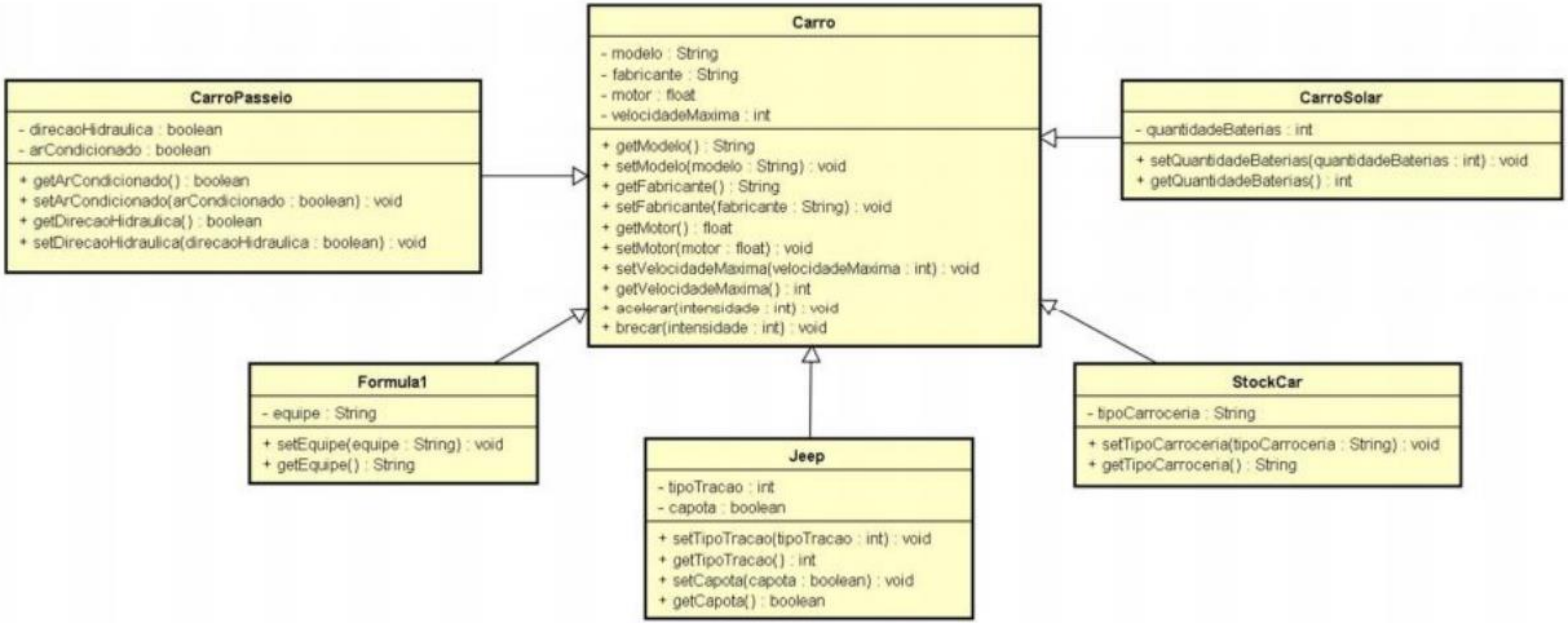
Representação no Diagrama de Classes



Exemplos de Herança



Representação no Diagrama de Classes



Como fazemos no código

Toda classe criada no C# é estendida a partir da classe **Object**.

A sinalização “ **:** ” é utilizada na declaração de uma classe para especificar quem é sua superclasse.

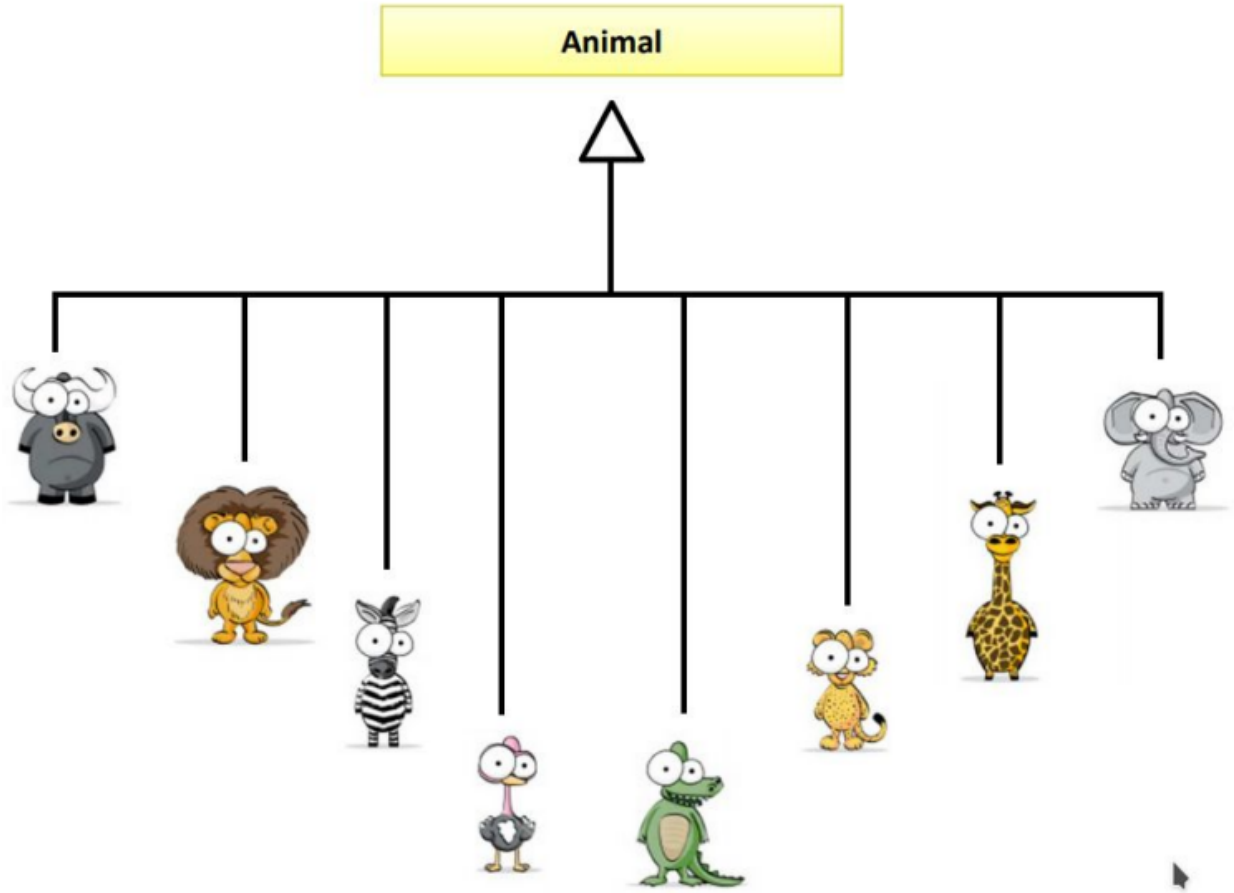
Caso a palavra-chave seja omitida, a classe **Object** será assumida como a superclasse da nova classe.

Como fazemos no código

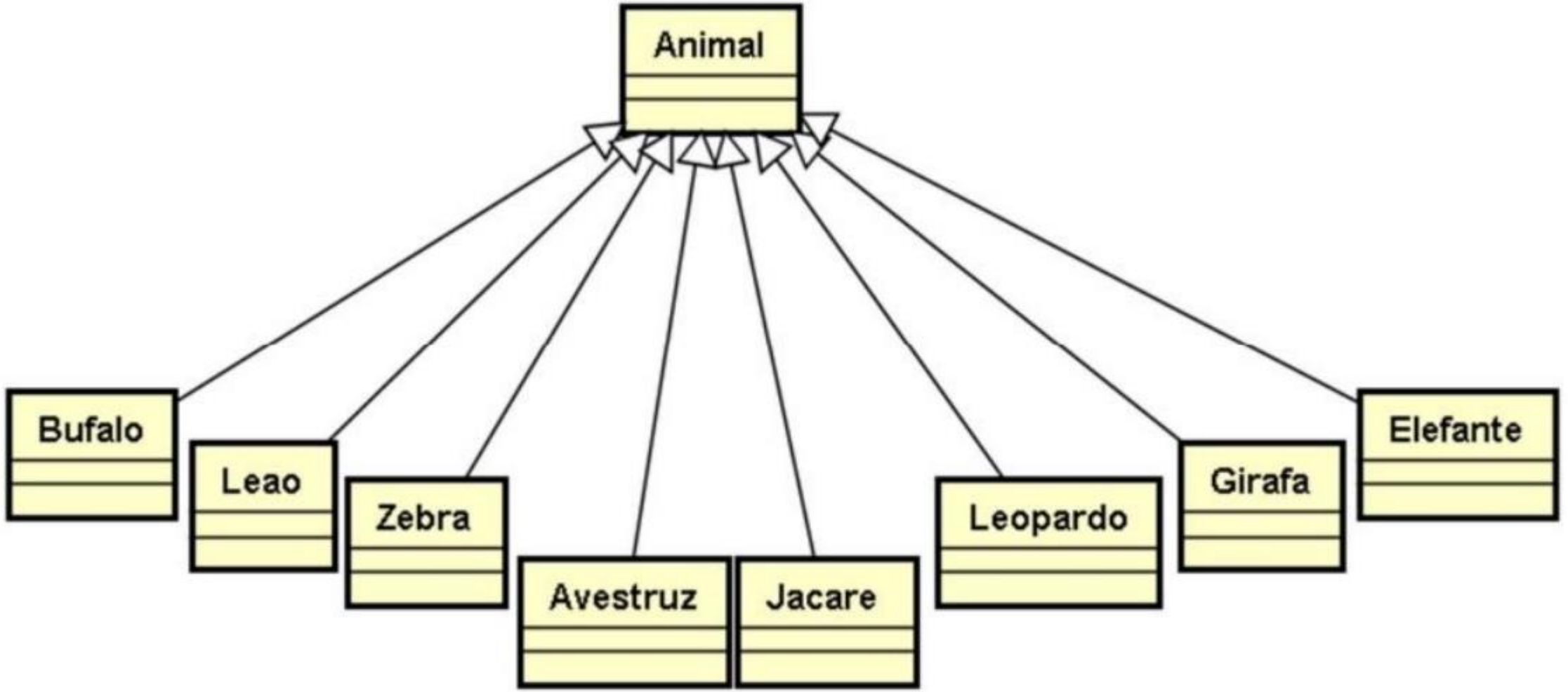
No exemplo exibido a seguir a classe **Formula1** estende a classe **Carro**:

```
0 referências
public class Formula1 : Carro
{
    0 referências
    public String Equipe { get; set; }
}
```

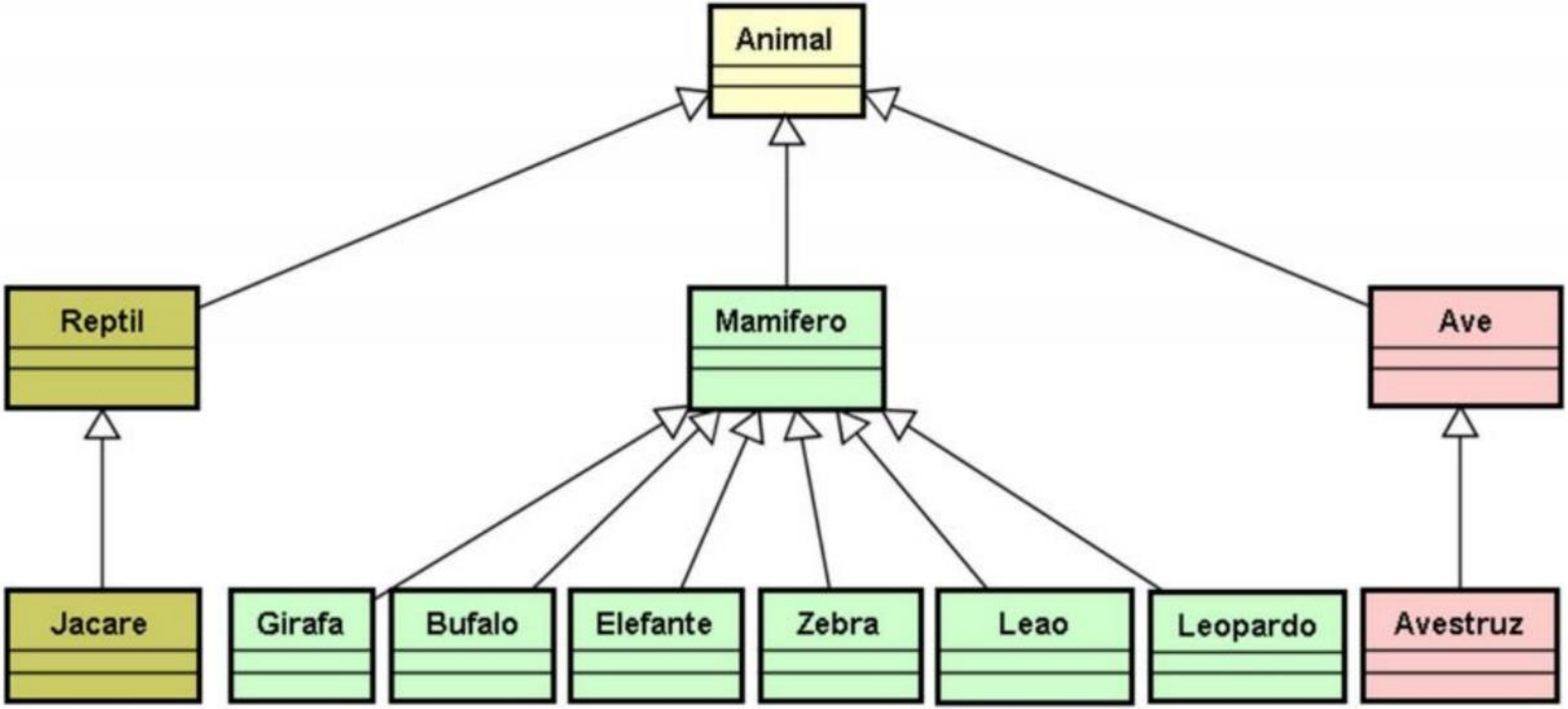
Exemplos de Herança



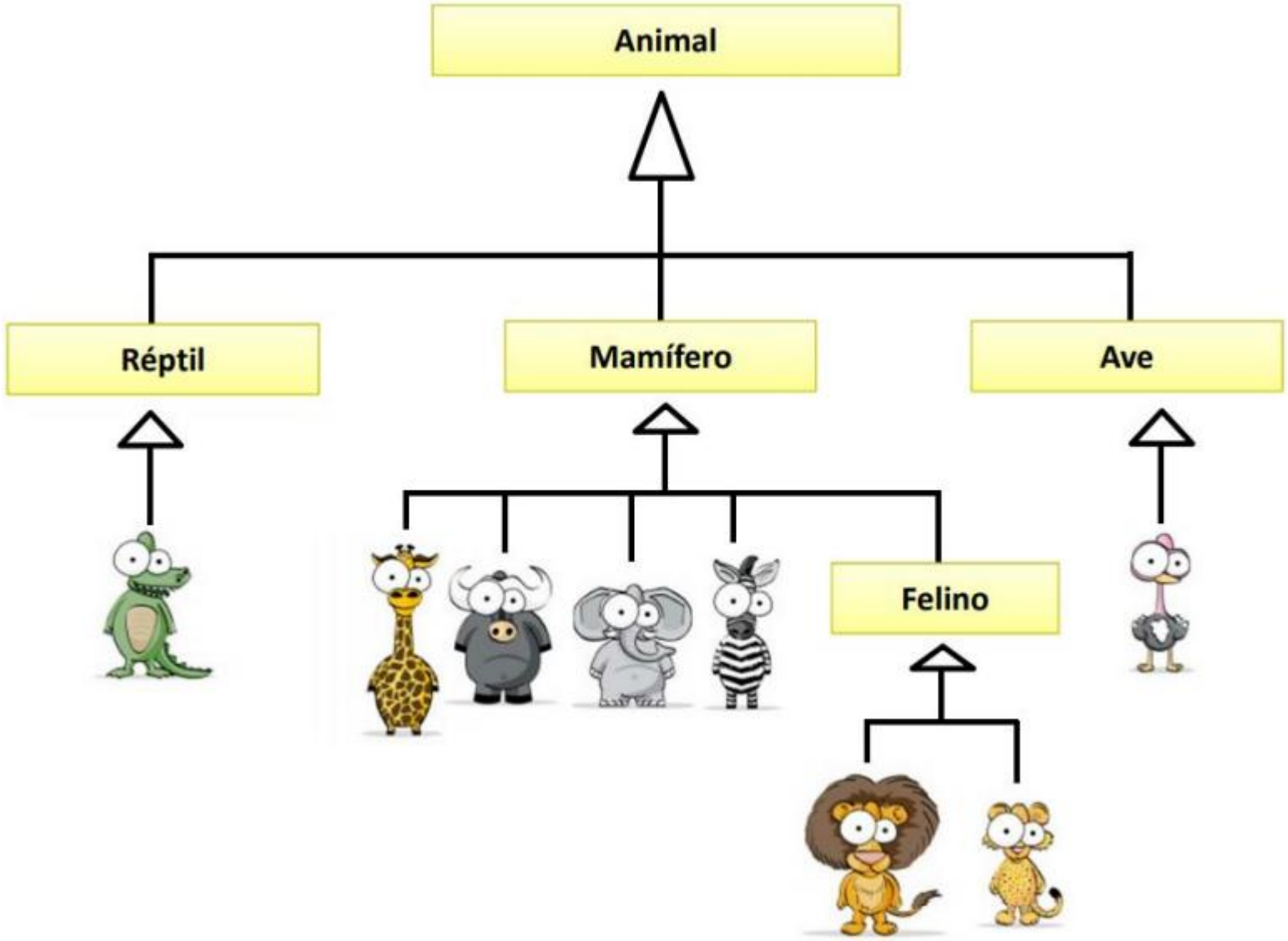
Exemplos de Herança



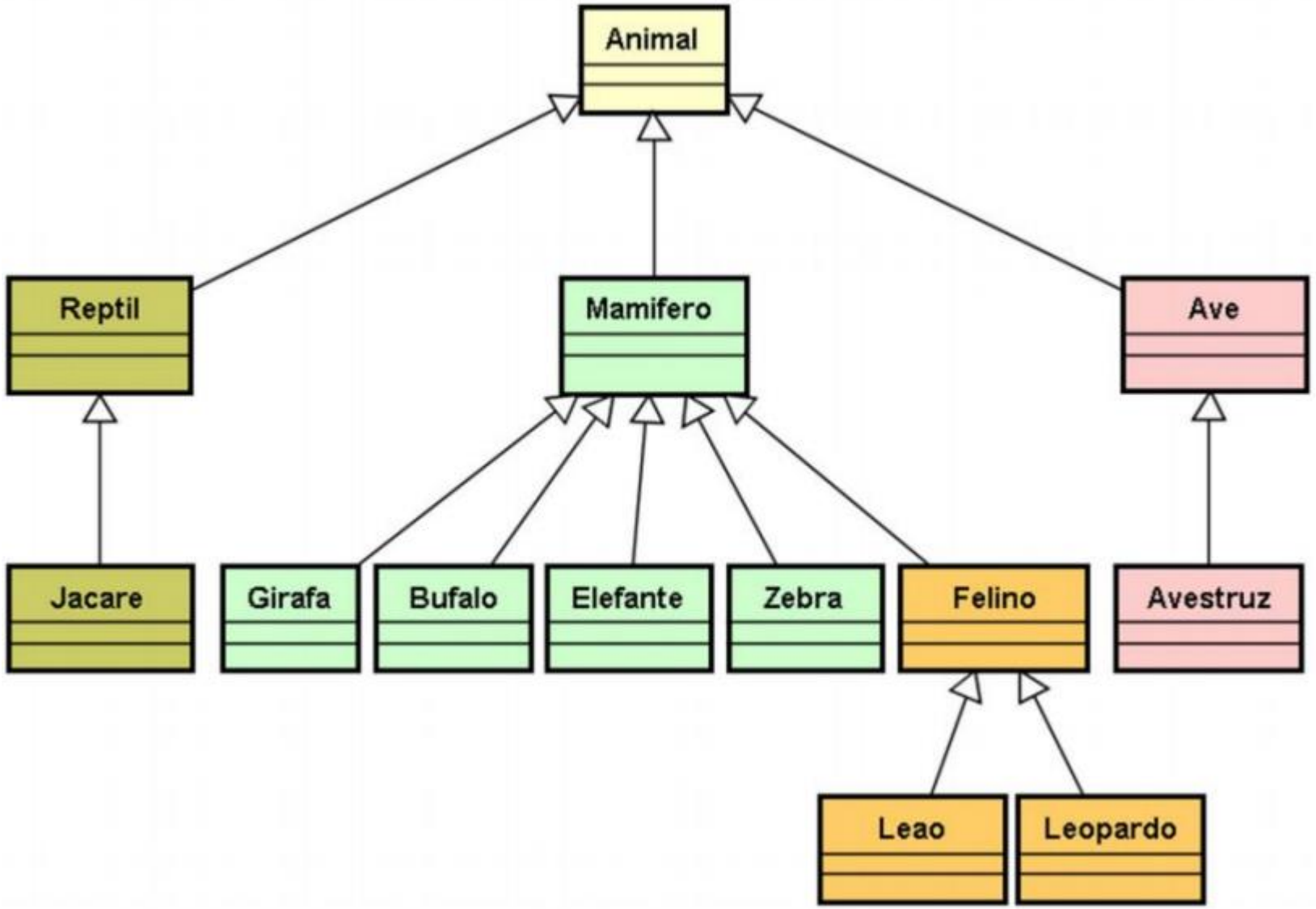
Exemplos de Herança



Exemplos de Herança



Exemplos de Herança



Exercício

Implemente em C# o seguinte diagrama de classes: