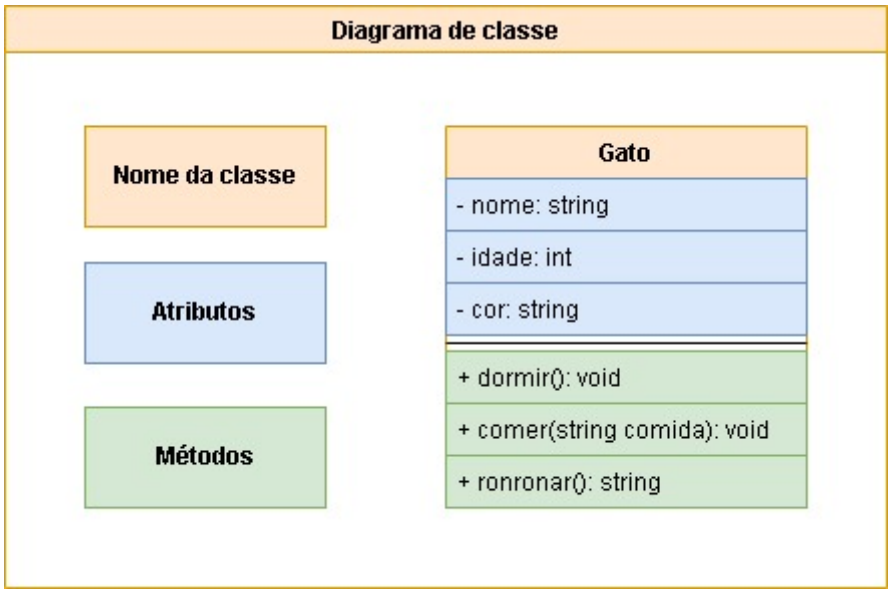
C#-10 - Polymorphism and inheritance

1. Diagrama de classe
2. Herança
3. Polimorfismo
4. Classes abstratas
5. Interfaces

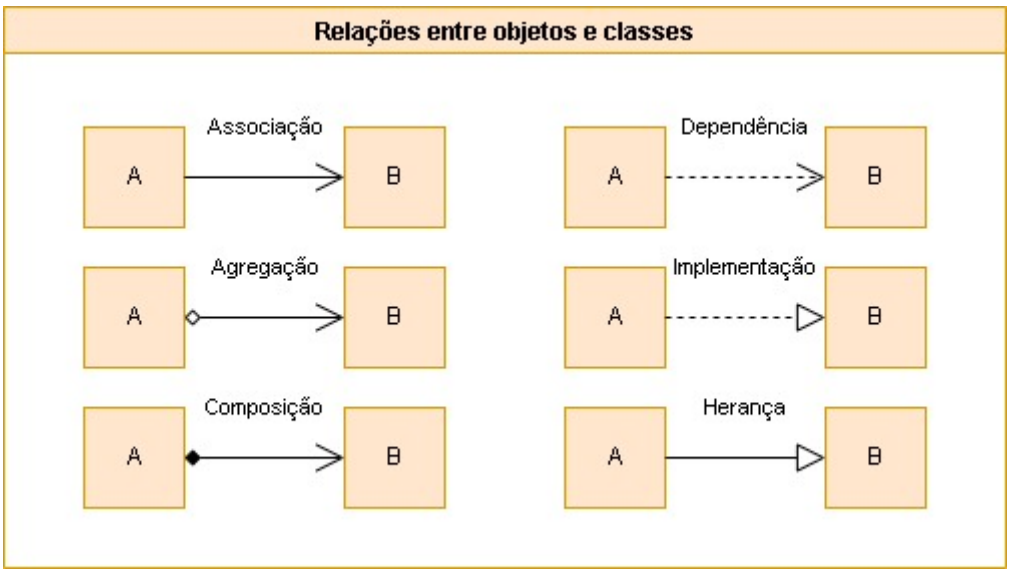
1. Diagrama de classe

É possível demostrar código de classes e interações utilizando **UML**(UML é a sigla do inglês Unified Modeling Language e significa Linguagem de Modelagem Unificada). Abaixo segue um pequeno exemplo de diagrama de classe:



Na representação acima é possível ver que uma classe possui alguns componentes como atributos e métodos, isso é muito importante para definir uma classe. Para desenvolver um diagrama é possível fazer gratuitamente no draw.io, no Link.

As classes podem se relacionar entre si e para isso é possível utilizar uma simbologia que será descrita abaixo:



Relação	Descrição
Dependência	Classe **A** pode ser afetada por mudanças na classe **B**;
Associação	Objeto **A** sabe sobre objeto **B**. Classe **A** depende de **B**;
Agregação	Objeto **A** sabe sobre objeto **B**, e consiste de **B**. Classe **A** depende de **B**;
Composição	Objeto **A** sabe sobre objeto **B**, consiste de **B**, e gerencia o ciclo de vida de **B**. Classe A depende de **B**;
Implementação	Classe **A** define métodos declarados na interface **B**. objetos de **A** podem ser tratados como **B**. Classe **A** depende de **B**;
Herança	Classe **A** herda a interface e implementação da classe **B** mas pode estende-lá. Objetos de **A** podem ser tratados como **B**. Classe A depende de **B**.

2. Herança

As classes dão suporte completo à herança, uma característica fundamental da programação orientada a objetos. Quando você cria uma classe, pode herdar de qualquer outra classe que não esteja definida como sealed , e outras classes podem herdar de sua classe e substituir métodos virtuais de classe. Além disso, você pode implementar uma ou mais interfaces.

A herança é realizada usando uma derivação, o que significa que uma classe é declarada usando uma classe base, da qual ela herda o comportamento e os dados. Uma classe base é especificada ao acrescentar dois-pontos e o nome de classe base depois do nome de classe derivada, dessa maneira:

```
public class Gato : Animal
{
    // Atributos, propriedades, construtores e métodos
}
```

Quando uma classe declara uma classe base, ela herda todos os membros da classe base, exceto os construtores.

Herança implícita

Todas as classes possuem uma Herança implícita com a classe **Object** que por si possui 8 membros em sua especificação:

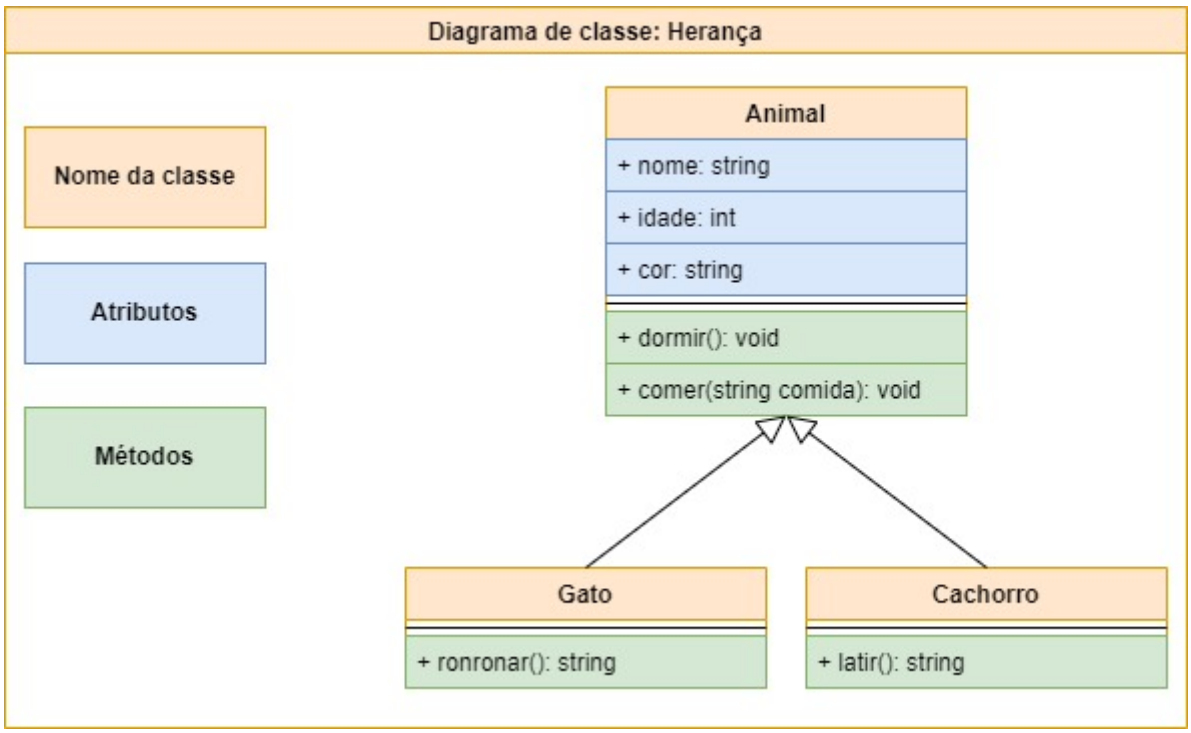
- **ToString** (Método): Método ToString público, que converte um objeto simples em sua representação de cadeia de caracteres, retorna o nome de tipo totalmente qualificado;
- **Equals** (Método): Métodos de teste de igualdade de dois objetos de instância publica;
- **Equals** (Método): Métodos de teste de igualdade de dois objetos de instância publica estática;
- **ReferenceEquals** (Método): Métodos de teste de igualdade de dois objetos de instância publica estática;
- **GetHashCode** (Método): calcula um valor que permite o uso de uma instância do tipo em conjuntos de hash;
- **GetType** (Método): Retorna um objeto **Type** que representa o tipo;
- **Finalize** (Método): Libera recursos não gerenciados antes que a memória de um objeto seja reivindicada pelo coletor de lixo;
- **MemberwiseClone** (Método): Cria um clone superficial do objeto atual.

Categoria do tipo	Herda implicitamente
class	Object
struct	ValueType, Object
enum	Enum, ValueType, Object
delegado	MulticastDelegate, Delegate, Object

3. Polimorfismo

O polimorfismo costuma ser chamado de o terceiro pilar da programação orientada a objetos, depois do encapsulamento e a herança. O polimorfismo é uma palavra grega que significa "de muitas formas" e tem dois aspectos distintos:

- Em tempo de execução, os objetos de uma classe derivada podem ser tratados como objetos de uma classe base, em locais como parâmetros de método, coleções e matrizes. Quando esse polimorfismo ocorre, o tipo declarado do objeto não é mais idêntico ao seu tipo de tempo de run-time;
- As classes base podem definir e implementar métodos virtuais e as classes *derivadas* podem substituí-lós, o que significa que elas fornecem sua própria definição e implementação.



4. Classes abstratas

O modificador abstract indica que o item que está sendo modificado tem uma implementação ausente ou incompleta. O modificador abstrato pode ser usado com classes, métodos, propriedades, indexadores e eventos. Use o modificador abstract em uma declaração de classe para indicar que uma classe se destina somente a ser uma classe base de outras classes, não instanciada por conta própria. Membros marcados como abstratos precisam ser implementados por classes não abstratas que derivam da classe abstrata.

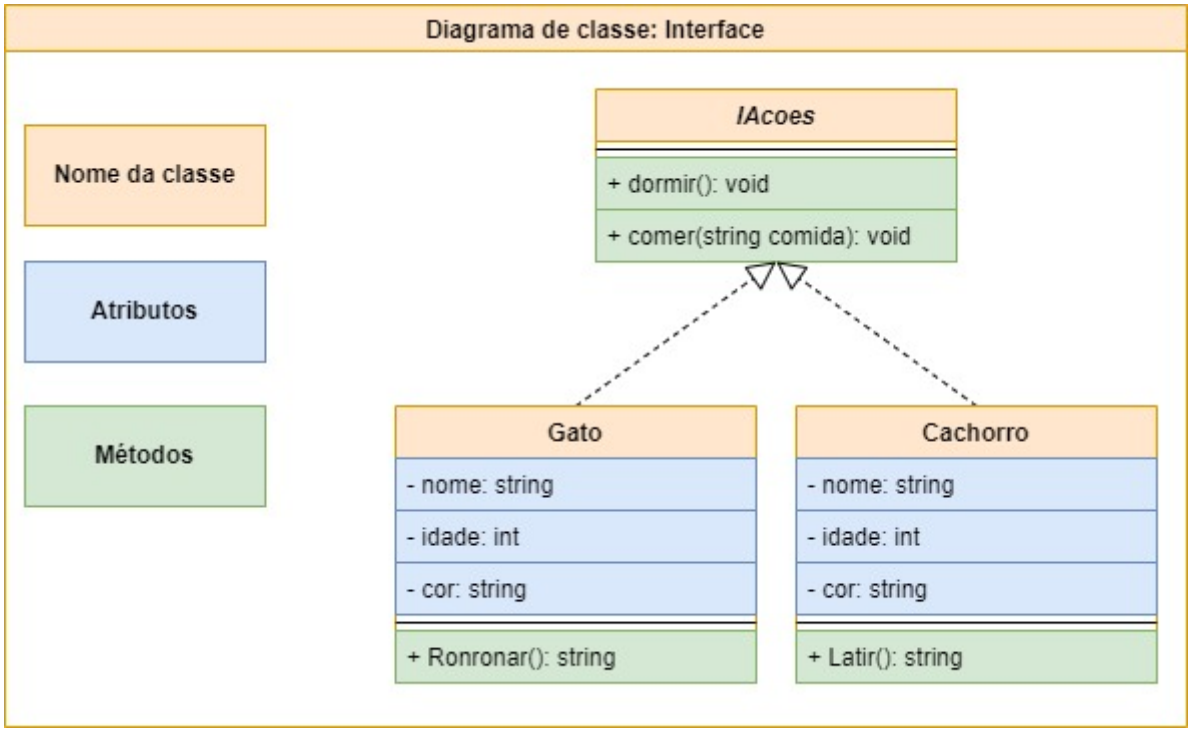
```
abstract class Animal
{
    public abstract string FazerBagunca();
}

class Cachorro : Animal
{
    // Sobreescrita do método FazerBagunca da Classe Animal
    public override string FazerBagunca() => "Comendo o sofá";

    static void Main()
    {
        var gato = new Cachorro();
        Console.WriteLine($"O cachorro esta {gato.FazerBagunca()}");
    }
}
// Output: O cachorro esta Comendo o sofá
```

5. Interfaces

Uma interface contém definições para um grupo de funcionalidades relacionadas que uma classe não abstrata ou um struct **deve implementar**. Usando interfaces, você pode, por exemplo, incluir o comportamento de várias fontes em uma classe. Essa funcionalidade é importante em C# porque a linguagem não dá suporte a várias heranças de classes.



O exemplo abaixo mostra como se definir uma interface:

```
interface IAcoes<T>
{
    void Dormir();
    void Comer(String comida);
}
```

Em seguida temos como fica sua implementação:

```
public class Gato : IAcoes<Gato>
{
    public string Make { get; set; }
    public string Model { get; set; }
    public string Year { get; set; }

    // Implementação da interface IAcoes<T>
    public void Dormir()
    {
         Console.WriteLine("O gato esta dormindo! ZzZz");
    }

    // Implementação da interface IAcoes<T>
    public void Comer(String comida)
    {
         Console.WriteLine("O gato esta comendo!");
    }
}
```