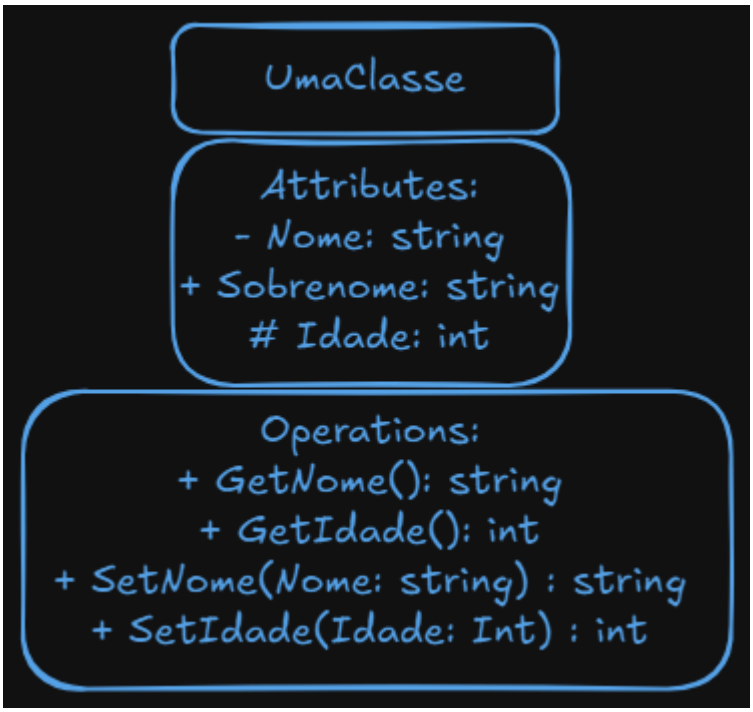
UML Básico

Classes

Classe representada em diagrama:



Modificadores de Acesso:

- `-` -> `private`
- `+` -> `public`
- `#` -> `protected`

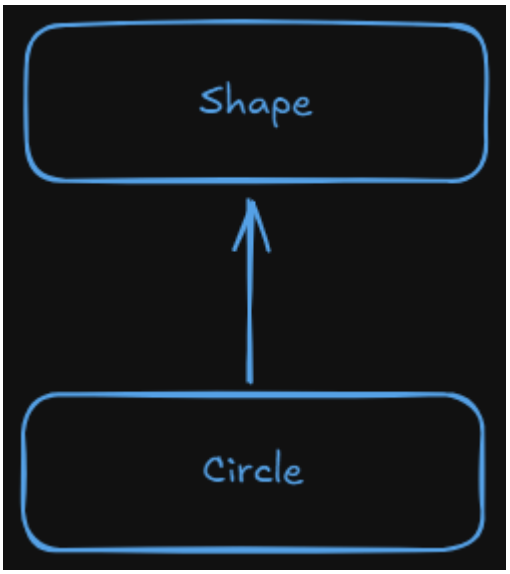
Classe representada em código (C#):

```
public class UmaClasse
{
   private string _nome;
   public string _sobrenome;
   protected int _idade;

   public UmaClasse(string nome, string sobrenome, int idade)
   {
      _nome = nome;
```

```
        _sobrenome = sobrenome;
        _idade = idade;
    }

    public string GetNome()
    {
        return _nome;
    }

    public int GetIdade()
    {
        return _idade;
    }

    public string SetNome(string nome)
    {
        _nome = nome;
        return _nome;
    }

    public int SetIdade(int idade)
    {
        _idade = idade;
        return _idade;
    }
}
```

Heranças

Herança representada em diagrama:

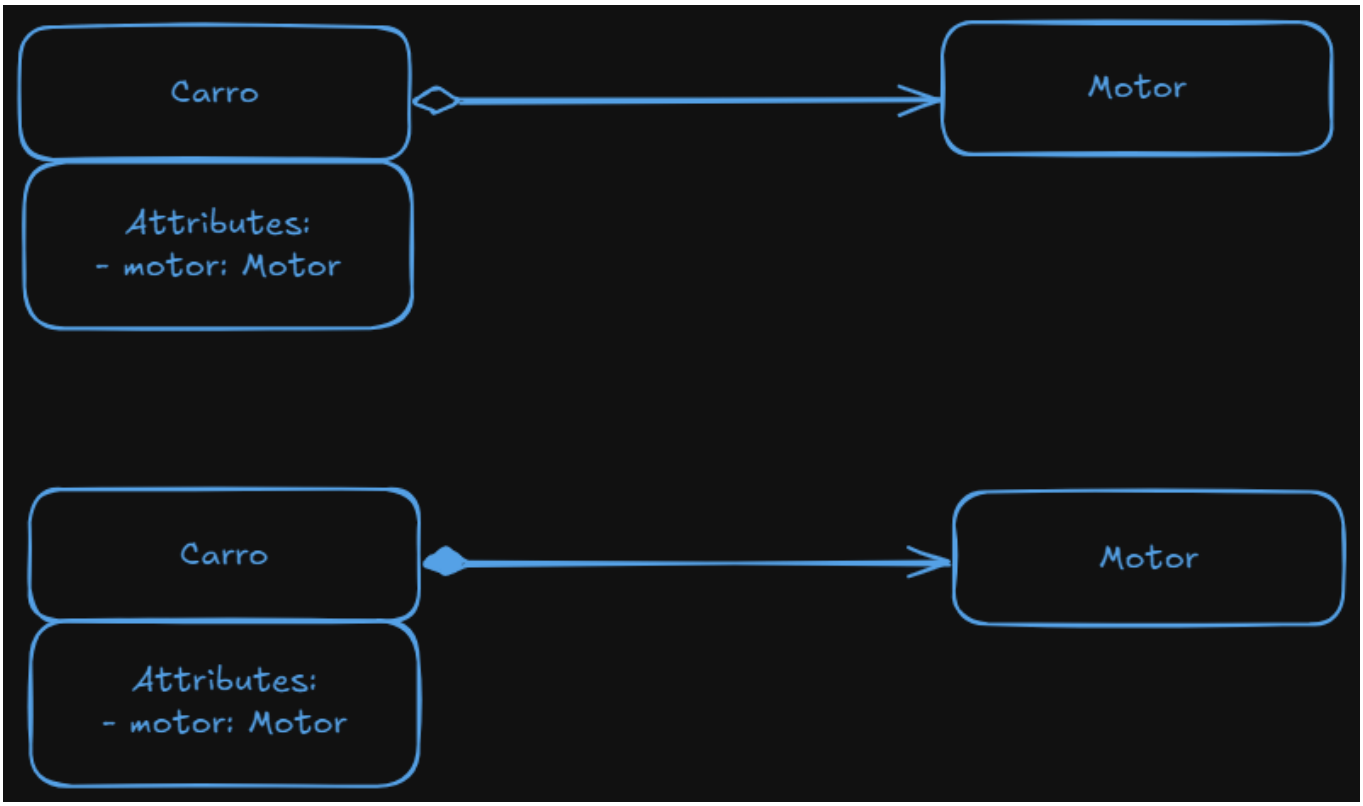


Herança representada em código (C#):

```
public class Shape
{
    // código omitido
}

public class Circle : Shape
{
    // código omitido
}
```

Agregação e Composição

Agregação e Composição representada em diagrama:

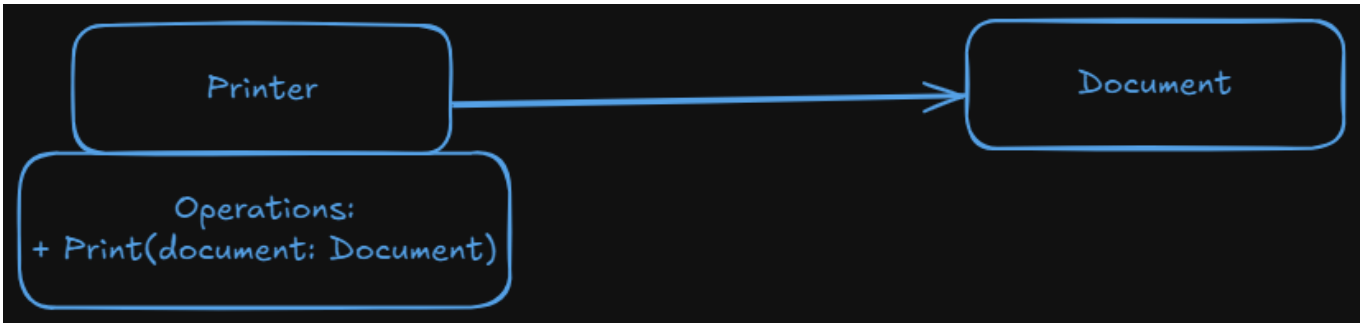


Agregação representada por código (C#):

```
public class Carro
{
    private Motor _motor;

    public Carro(Motor motor)
    {
        _motor = motor;
    }
}

public class Motor
{
    // codigo emitido
}

const motor = new Motor();
const carro = new Carro(motor);

Console.WriteLine(carro);
```

Dependência

Dependência representada por diagrama:

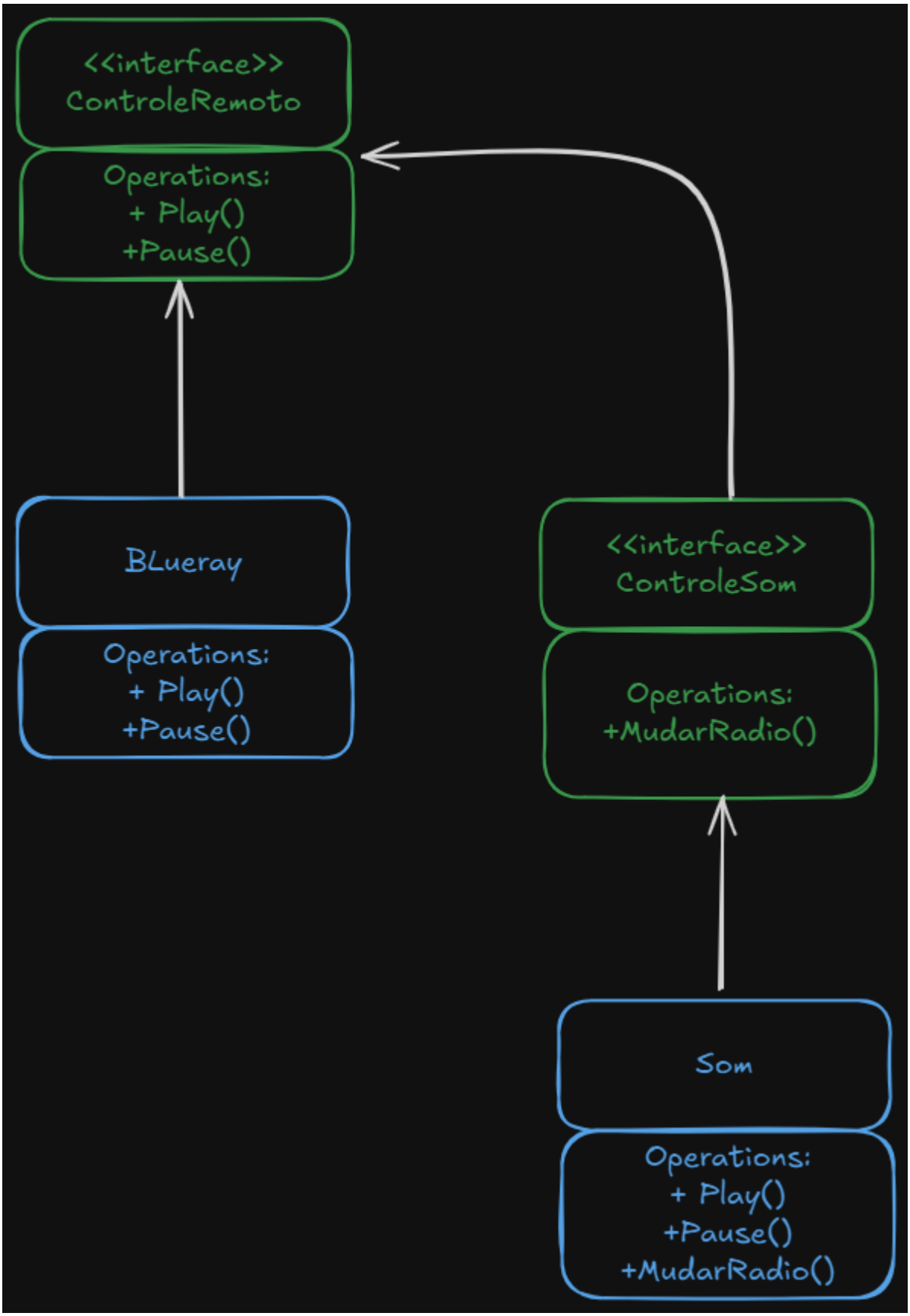


Dependência representada por código (C#):

```
public class Printer
{
    public void Print(Document document)
    {
        Console.WriteLine($"Printing document: {document}");
    }
}

public class Document
{
    // código omitido
}
```

Interface

Interface representada por diagrama:



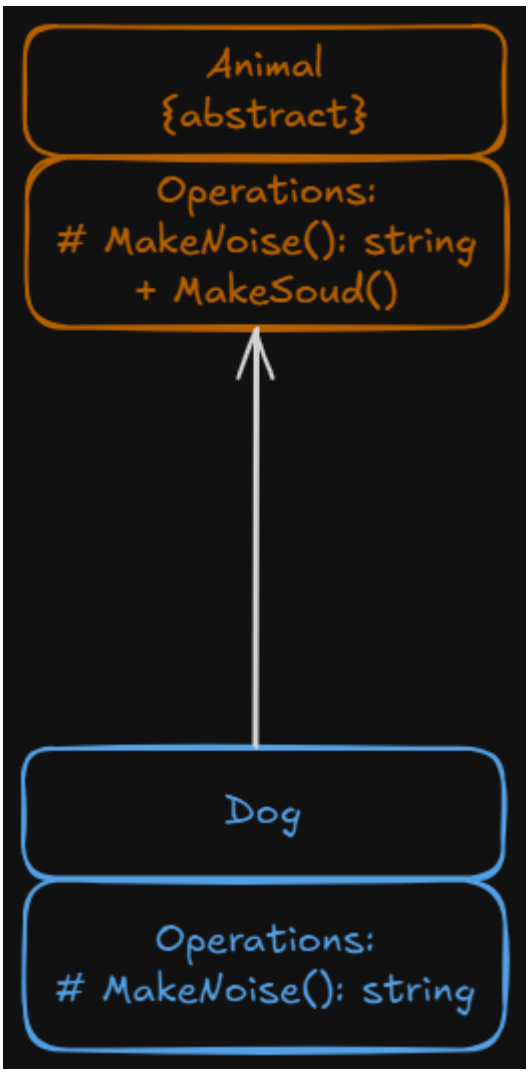
Interface representada por código (C#):

```
public interface IControleRemoto
{
    public void Play();
    public void Pause();
}

public interface IControleSom : IControleRemoto
{
    public void MudarRadio();
}

public class Blueray : IControleRemoto
{
    public void Pause()
    {
        throw new NotImplementedException();
    }

    public void Play()
    {
        throw new NotImplementedException();
    }
}

public class Som : IControleSom
{
    public void MudarRadio()
    {

        throw new NotImplementedException();
    }

    public void Pause()
    {
        throw new NotImplementedException();
    }

    public void Play()
    {
        throw new NotImplementedException();
```

```
    }
}
```

Abstração

Abstração representada em diagrama:



Abstração representada em código(C#):

```
public abstract class Animal
{
    protected abstract string MakeNoise();

    public void MakeSound()
    {
        Console.WrileLine(this.MakeNoise());
    }
```

```
}

public class Dog : Animal
{
    protected override string MakeNoise()
    {
        return "Woof!";
    }
}
```