

A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the top-left corner of the slide.

# AceLeraDev C#

Programação Estruturada e Orientada a Objetos

Módulo 2

A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the bottom-right corner of the slide.



# Tópicos desta aula:

Tópico 1: Programação Estruturada

Tópico 2: Programação Orientada a Objetos



# Programação Estruturada

- É uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e iteração.
- Orienta os programadores para a criação de estruturas simples nos programas, usando as sub-rotinas e as funções. Foi à forma dominante na criação de software anterior à programação orientada por objetos.

# Programação Estruturada

- Sequência: são implementados os passos de processamento necessários para descrever determinada funcionalidade. Um exemplo básico seria um fluxograma, onde primeiro é executado a Etapa 1 e após a sua conclusão a Etapa 2 é executada, e assim por diante.
- Seleção: o fluxo a ser percorrido depende de uma escolha, se a Condição 1 for verdade faça Processamento 1, caso contrário, se a Condição 2 for verdade faça Processamento 2, caso contrário, se a Condição 3 for verdade faça Processamento 3, e assim por diante.

# Programação Estruturada

- Iteração: é permito a execução de instruções de forma repetida, onde ao fim de cada execução a condição é reavaliada e enquanto seja verdadeira a execução de parte do programa continua.
- Modularização: a medida que o sistema vai tomando proporções maiores, é mais viável que o mesmo comece a ser dividido em partes menores. Essa técnica ficou conhecida como Subprogramação ou Modularização. No desenvolvimento utilizamos essa técnica através de procedimentos, funções, métodos, rotinas e uma série de outras estruturas.

# Programação Orientada a Objetos

- é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos'.
- A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem **Orientada a Objetos**. Como estamos lidando com uma representação de um objeto real, temos que imaginar o que esse objeto irá realizar dentro de nosso sistema.

# Programação Orientada a Objetos

- A orientação a objetos permite que haja uma reutilização do código criado, isso é possível devido ao fato de que as linguagens orientada a objetos trazem representações muito claras de cada um dos elementos, e esses elementos normalmente não são interdependentes. Essa independência entre as partes do software é o que permite que esse código seja reutilizado em outros sistemas no futuro.
- Os principais conceitos do paradigma orientado a objetos são Classes e Objetos, Encapsulamento, Herança e Polimorfismo.



# Revisão do que vimos hoje:

Tópico 1: Programação Estruturada

Tópico 2: Programação Orientada a Objetos



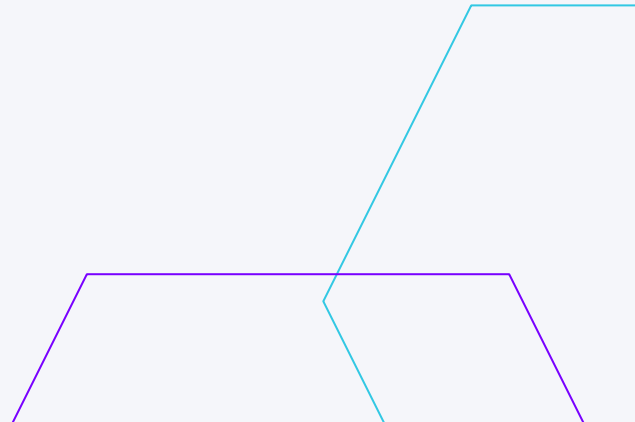




# Tópicos da próxima aula:

Tópico 1: Objetos

Tópico 2: Classe





# AceLeraDev C#

Objetos e Classe

Módulo 2

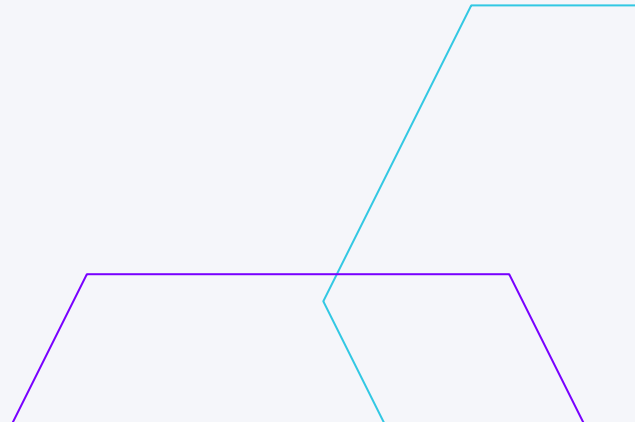




# Tópicos desta aula:

Tópico 1: Objetos

Tópico 2: Classe



# Objetos

- É a abstração do mundo real para o mundo virtual.
- Tudo é objeto ! Uma casa, um carro, uma pessoa ...
- É algo distinguível que possui características (atributos) e comportamentos (métodos).

# Objetos

## Características:

- Mulher
- Ruiva
- Usa Óculos vermelho
- Olhos castanhos
- Dev C#



## Comportamentos:

- Programar
- Estudar
- Falar
- Tomar café
- Comer chocolate

# Classe

- É o agrupamento de objetos com a mesma estrutura de dados e comportamentos.
- É uma descrição que abstrai um conjunto de **objetos** com características similares.
- É um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por **objetos**.

# Classe

Produto
- codigo : int - nome : string - descricao : string - preco_compra : decimal - preco_venda : decimal - quantidade_estoque : int - ativo : bool - data_cadastro : DateTime
+ inserir() : void + alterar() : void + excluir() : void + pesquisar() : void

Nome da classe

Atributos

Métodos

## Classe Cães

### Objetos cachorros



BOB

Anda  
Fala  
Come  
Dorme  
PegaOsso



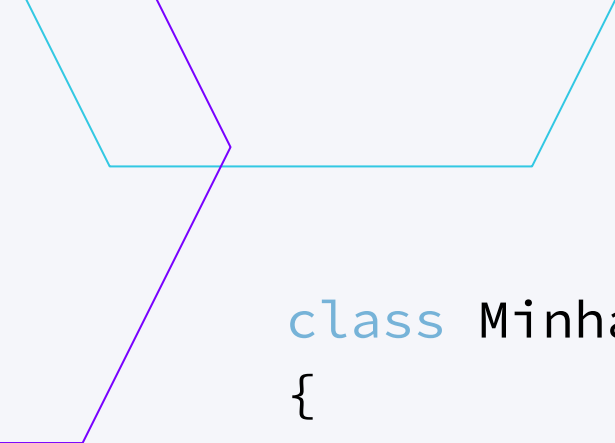
SCOOBY

Anda  
Fala  
Come  
Dorme  
PegaOsso



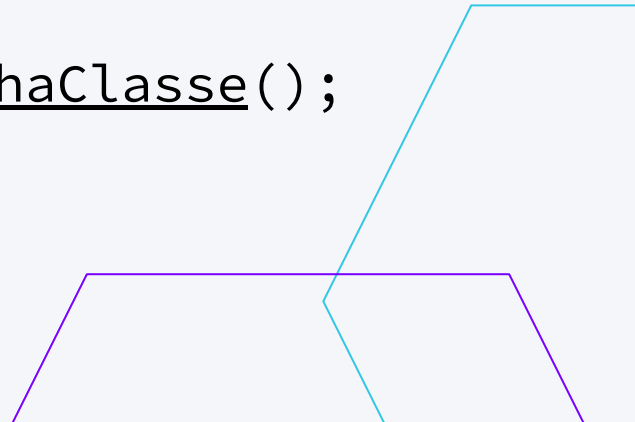
REX

Anda  
Fala  
Come  
Dorme  
PegaOsso



```
class MinhaClasse
{
    // o código da minha classe
}
```

```
MinhaClasse meuObjeto = new MinhaClasse();
```







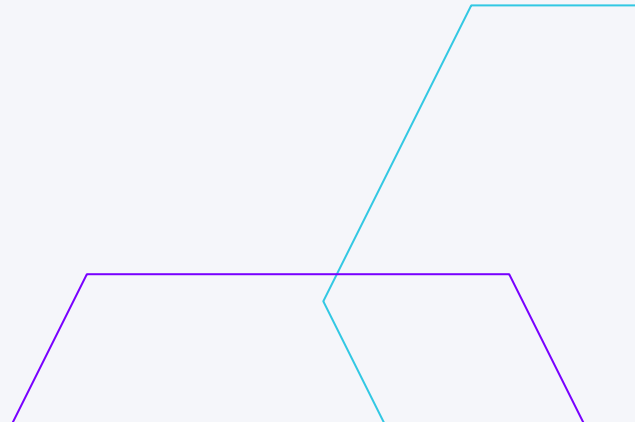
**TALK IS CHEAP  
SHOW ME  
THE CODE**



# Revisão do que vimos hoje:

Tópico 1: Objetos

Tópico 2: Classe





# Tópicos da próxima aula:

Tópico 1: Herança

Tópico 2: Polimorfismo



A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the top-left corner of the slide.

# AceLeraDev C#

Herança e Polimorfismo

Módulo 2

A series of overlapping geometric shapes, primarily triangles and quadrilaterals, in shades of teal and purple, located in the bottom-right corner of the slide.



# Tópicos desta aula:

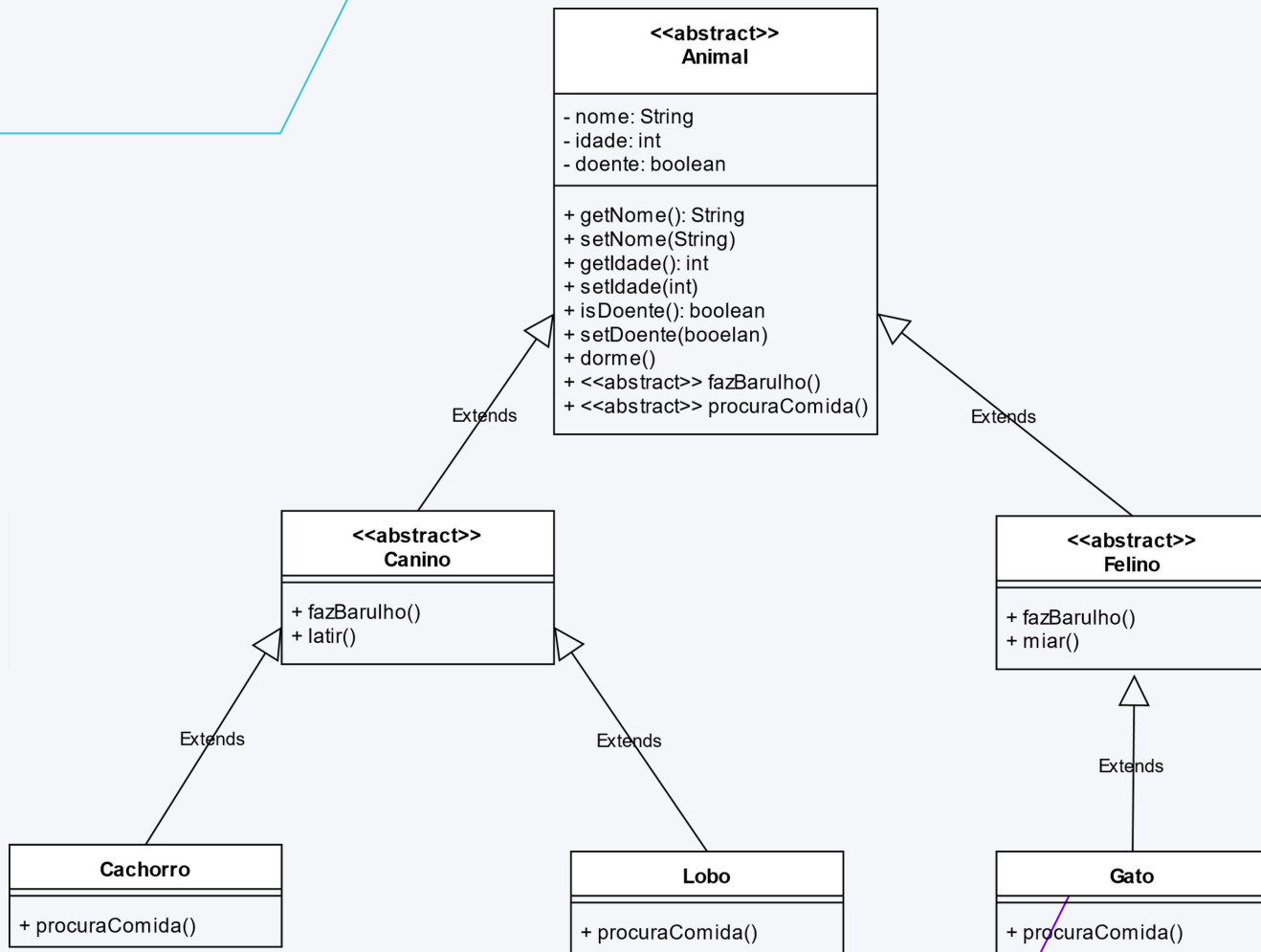
Tópico 1: Herança

Tópico 2: Polimorfismo



# Herança

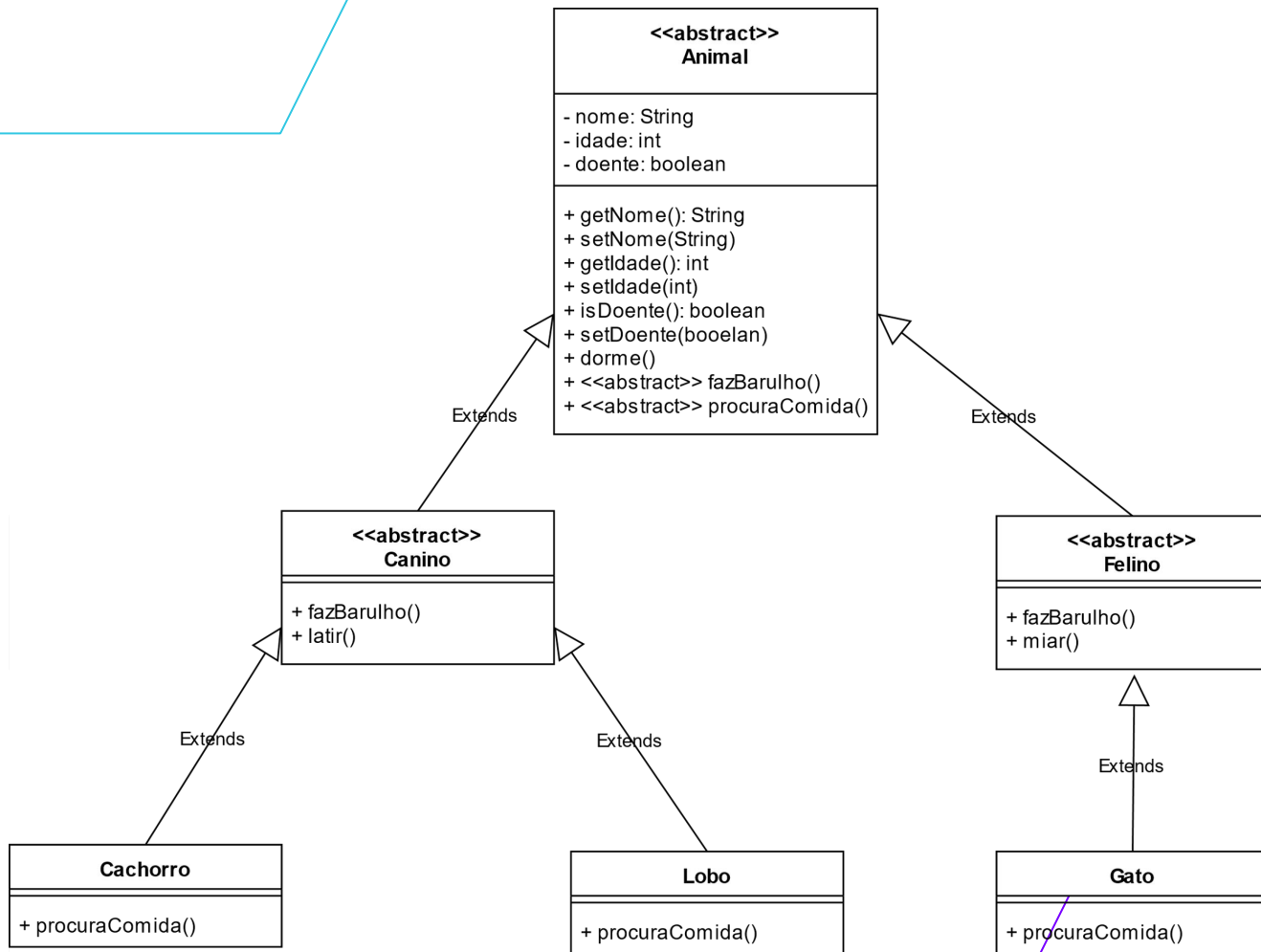
- Iniciando o princípio de reuso, podemos utilizar herança para evitar reescrever código. A Classe pai possui métodos e propriedades que todas as classes filhas possuem e as filhas possuem apenas as específicas a ela.
- A Herança possibilita que as classes compartilhem seus atributos, métodos e outros membros da classe entre si. Para a ligação entre as classes, a herança adota um relacionamento esquematizado hierarquicamente

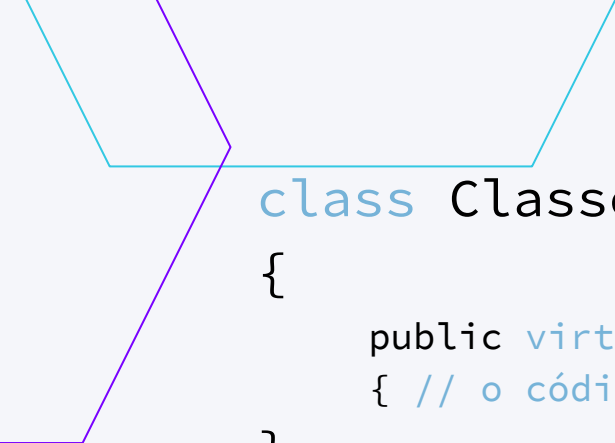


# Polimorfismo

- Definimos **Polimorfismo** como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, **embora apresentem a mesma assinatura, comportam-se de maneira diferente** para cada uma das classes derivadas.
- Permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam.





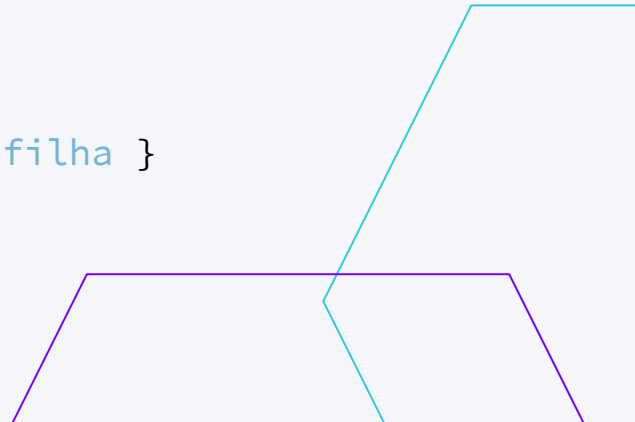


```
class ClasseMae
```

```
{  
    public virtual void MeuMetodo()  
    { // o código do meu método }  
}
```

```
class ClasseFilha : ClasseMae
```

```
{  
    public override void MeuMetodo()  
    { base.MeuMetodo(); // código da classe filha }  
}
```





**TALK IS CHEAP  
SHOW ME  
THE CODE**

The image features a central graphic with the text "TALK IS CHEAP SHOW ME THE CODE" in a bold, white, sans-serif font. The text is arranged in three lines and is set against a thick, black, hand-drawn style outline. This entire graphic is placed on a light gray rectangular background. The overall composition is set on a light blue background, which is decorated with faint, thin-lined geometric shapes in purple and teal, including triangles and polygons, primarily located in the top-left and bottom-right corners.



# Revisão do que vimos hoje:

Tópico 1: Herança

Tópico 2: Polimorfismo





# Tópicos da próxima aula:

Tópico 1: Encapsulamento

Tópico 2: Modificadores de acesso



# AceLeraDev C#

Encapsulamento e Modificadores de acesso

Módulo 2



# Tópicos desta aula:

Tópico 1: Encapsulamento

Tópico 2: Modificadores de acesso



# Encapsulamento

- É a técnica utilizada para esconder uma ideia.
- É a técnica que faz com que detalhes internos do funcionamento dos métodos de uma classe permaneçam **ocultos** para os objetos.
- O princípio de esconder a estrutura de dados utilizada e somente prover uma interface bem definida.



# Encapsulamento



# Modificadores de acesso

- `public`: O acesso não é restrito.
- `private`: O acesso é limitado ao tipo recipiente.
- `protected`: O acesso é limitado à classe que os contém ou aos tipos derivados da classe que os contém.
- `internal`: O acesso é limitado ao assembly atual.

# Modificadores de acesso

- `protected internal`: O acesso é limitado ao assembly atual ou aos tipos derivados da classe que os contém.
- `private protected`: O acesso é limitado à classe que o contém ou a tipos derivados da classe que o contém no assembly atual. Disponível desde o C# 7.2.

The background features abstract geometric lines in teal and purple. In the top-left corner, a teal line forms a horizontal segment, while a purple line descends diagonally. In the bottom-right corner, a teal line forms a horizontal segment, and a purple line ascends diagonally. These lines intersect to create a series of overlapping trapezoidal shapes.

```
public class MinhaClasse  
{  
    // o código da minha classe  
}
```



**TALK IS CHEAP  
SHOW ME  
THE CODE**

The image features a central graphic with the text "TALK IS CHEAP SHOW ME THE CODE" in a bold, white, sans-serif font. The text is arranged in three lines and is set against a thick, black, hand-drawn style outline that has a slightly irregular, cloud-like shape. This entire graphic is placed on a light gray rectangular background. The overall composition is set against a light blue background. In the top-left corner, there are thin, intersecting lines in purple and teal. In the bottom-right corner, there are similar thin lines in teal and purple, forming a partial geometric pattern.



# Revisão do que vimos hoje:

Tópico 1: Encapsulamento

Tópico 2: Modificadores de acesso





# Tópicos da próxima aula:

Tópico 1: Classe Abstrata

Tópico 2: Classe Estática

Tópico 3: Construtores



# AceLeraDev C#

Classe Abstrata, Estática e construtor

Módulo 2



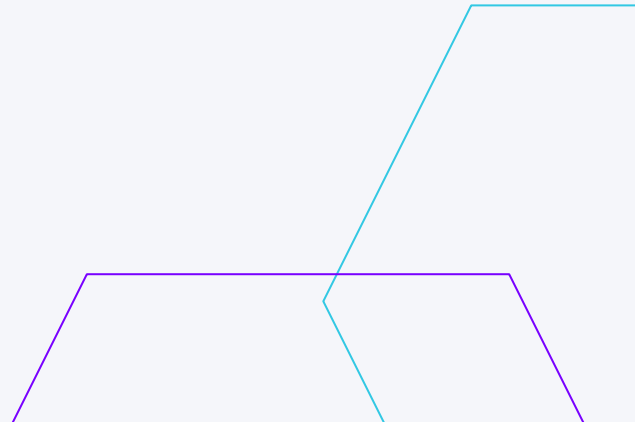


# Tópicos desta aula:

Tópico 1: Classe Abstrata

Tópico 2: Construtores

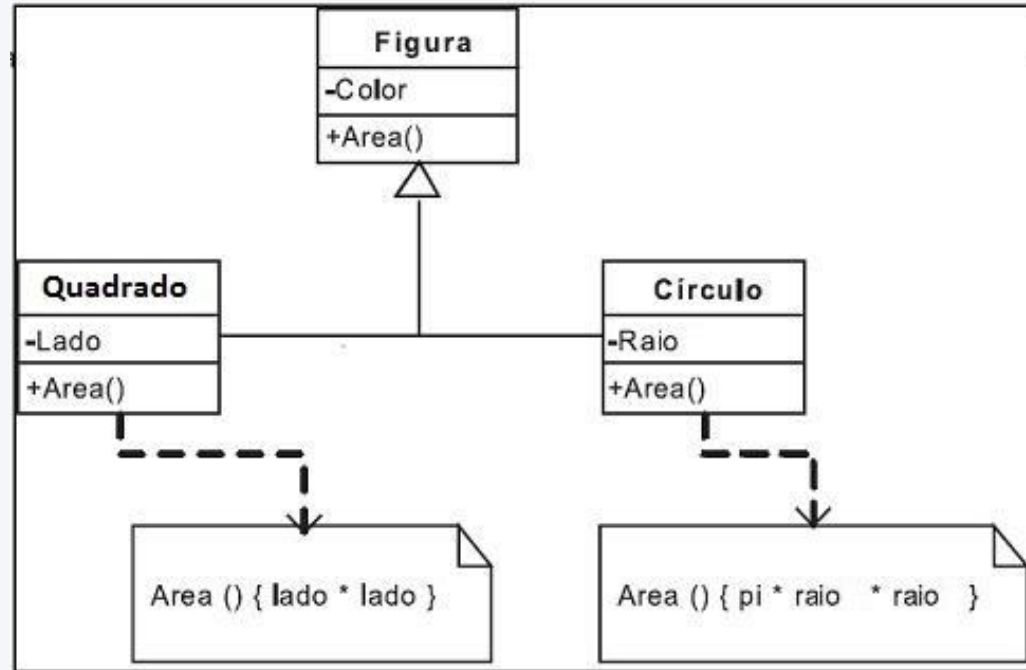
Tópico 3: Classe Estática



# Classe abstrata

- As classes abstratas são as que não permitem realizar qualquer tipo de instância. São classes feitas especialmente para serem modelos para suas classes derivadas.

# Classe abstrata





```
public abstract class MinhaClasse
{
    // o código da minha classe
}
```

# Construtor

- É um método que é executado sempre que uma classe é instanciada.
- Uma classe pode ter vários construtores que usam argumentos diferentes. Os construtores permitem que o programador defina valores padrões, limite a instanciação e escreva códigos flexíveis e fáceis de ler.
- Um construtor é um método cujo nome é igual ao nome do seu tipo. Sua assinatura de método inclui apenas o nome do método e sua lista de parâmetros, não inclui um tipo de retorno



```
public class MinhaClasse
```

```
{
```

```
    public MinhaClasse()
```

```
    { // código }
```

```
    public MinhaClasse(string param1)
```


```
    { // código }
```

```
}
```



# Classe Estática

- Não podemos usar a palavra-chave `new` para criar instâncias de uma **classe estática** já que, elas não podem ser instanciadas. Além disso, elas não podem ter construtores.
- Uma classe estática pode ser usada como um contêiner conveniente para conjuntos de métodos que apenas operam nos parâmetros de entrada e não precisam obter ou definir nenhum campo interno da instância.

The image features decorative geometric lines in purple and teal. In the top-left corner, a purple line and a teal line form a series of connected segments. In the bottom-right corner, similar purple and teal lines form another set of geometric shapes, including a hexagon-like structure.

```
public static class MinhaClasse  
{  
    // o código da minha classe  
}
```





**TALK IS CHEAP  
SHOW ME  
THE CODE**



# Revisão do que vimos hoje:

Tópico 1: Classe Abstrata

Tópico 2: Classe Estática

Tópico 3: Construtores

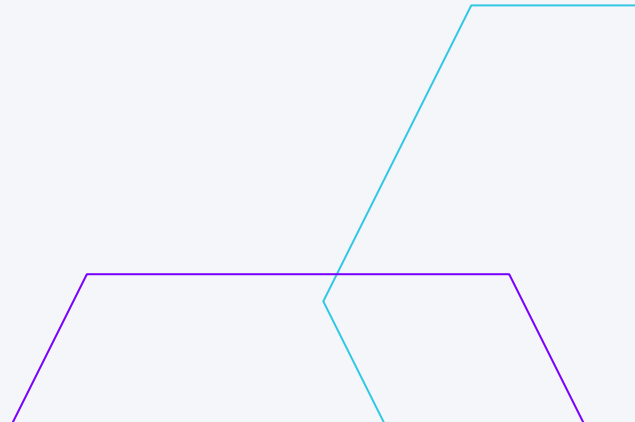




# Tópicos da próxima aula:

Tópico 1: Interface

Tópico 2: Namespace





# AceLeraDev C#

Interface e Namespace

Módulo 2

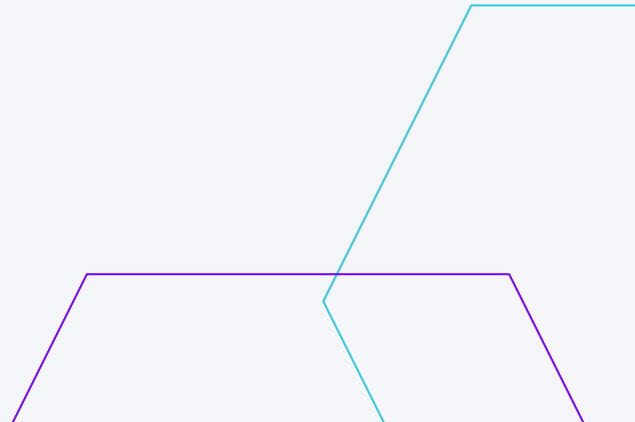




# Tópicos desta aula:


Tópico 1: Interface

Tópico 2: Namespace



# Interface

- Podemos definir como interface o **contrato** entre a classe e o mundo exterior. Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por esta interface.
- Uma interface define um contrato. Qualquer classe que implemente esse contrato deve fornecer uma implementação dos membros definidos na interface.

The image features decorative geometric lines in teal and purple. In the top-left corner, a teal line forms a partial hexagon, and a purple line extends diagonally downwards. In the bottom-right corner, a teal line forms another partial hexagon, and a purple line extends diagonally upwards. These lines are thin and serve as a background element for the code snippet.

```
public interface IMinhaInterface
{
    // o código da minha interface
}
```

The slide features decorative geometric lines in purple and teal. In the top-left corner, a purple line forms a large 'V' shape, and a teal line forms a smaller 'V' shape nested within it. In the bottom-right corner, there are two overlapping 'V' shapes, one in teal and one in purple.

# Namespace

- É semelhante ao conceito de uma pasta do sistema de arquivos em um computador.
- Ele agrupa classes e tipos por semântica e é declarado com a palavra-chave namespace.





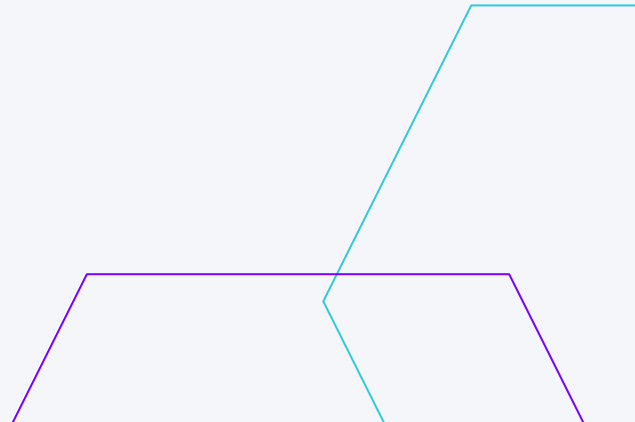
**TALK IS CHEAP  
SHOW ME  
THE CODE**



# Revisão do que vimos hoje:

Tópico 1: Interface

Tópico 2: Namespace





# Tópicos da próxima aula:

Tópico 1: Generics

Tópico 2: Extension Methods

Tópico 3: Exceptions

