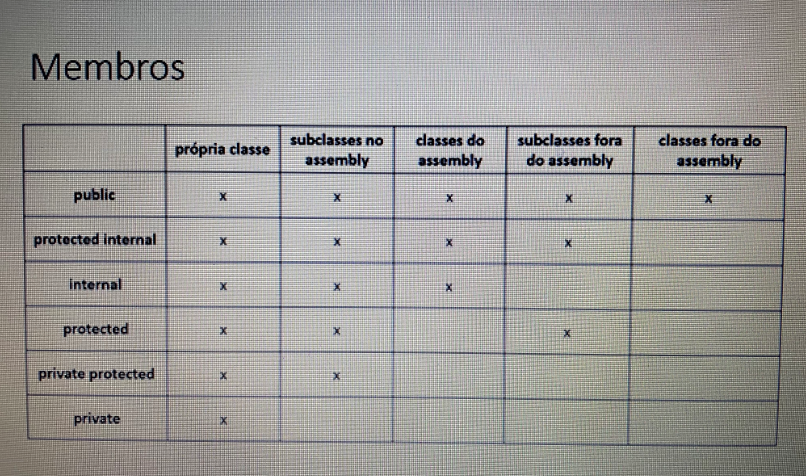
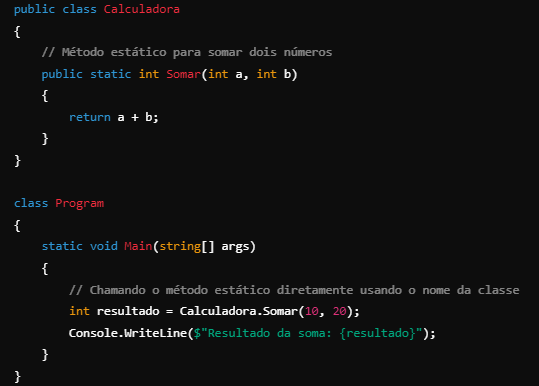
**CONCEITOS DIVERSOS DE C# - Ao final do curso organizar tudo**

* ***Upcastin/Downcasting:***
* ***Classe/métodos selados:***
* ***Vetor/Lista tipo Struct e Classe:***
* ***Categorias de classes:***
* ***Modificadores de acesso:***



* ***Métodos estáticos e não estáticos:***
  + ***Método não estático:*** é quandoo comportamento do método varia de acordo com os atributos de cada instância. Depende dos atributos da instância. **Esse método pertence a uma instância da classe!**
  + ***Método estático:*** é quando o método não depende de uma instância da classe. Chamamos a classe, o método, e passamos os parâmetros. Não depende de nenhum atributo para ser chamado, não precisa de instância. **Esse método pertence a classe em si!**



* ***ToString():***
* ***O que é:*** O método ToString() em C# é uma maneira poderosa e comum de converter um objeto em uma representação de string.
* ***Como funciona:*** O método ToString() está presente em todas as classes do C# porque é definido na classe base Object, da qual todas as classes derivam. Ele retorna uma descrição simples do objeto. Se o objeto for um valor numérico, ele retorna o número em formato de string. Se for uma instância de classe personalizada, ele retorna o nome completo da classe, a menos que você sobrescreva o método.
* ***Para que serve:***

- Quando você precisa exibir dados de uma variável ou objeto de forma textual;

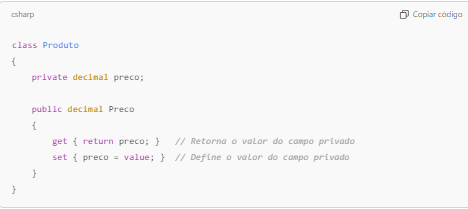
- Para converter qualquer valor ou objeto em uma string;

- Quando você cria suas próprias classes e deseja que o objeto tenha uma representação em string mais legível e informativa ao ser impresso;

- Usar ToString() com parâmetros para formatar a saída da string (números, datas, moedas, etc.);

- Entre outros;

* ***Sobrecarga:***
* ***This:***
* ***Get e Set(colocar na parte de encapsulamento também):***
  + ***O que é:*** getters e setters são partes das propriedades, que fornecem uma maneira de acessar e modificar os atributos de uma classe de forma controlada. Eles são usados para encapsular campos privados, permitindo que você controle como os dados são acessados ou alterados.
  + ***Estrutura Básica de uma Propriedade com Get e Set:***
* get: Retorna o valor de um campo privado.
* set: Atribui um valor ao campo privado, permitindo a inclusão de validações ou lógica adicional durante a atribuição.
  + ***Quando usar:***
* Encapsulamento de dados: Sempre que você tiver atributos que precisem ser controlados ou validados, getters e setters são uma ótima escolha.
* Validação de entrada: Se o valor de um atributo precisar seguir certas regras, como não ser nulo ou estar dentro de um intervalo específico, utilize um set para incluir a validação.
* Lógica no acesso: Quando você precisar realizar alguma operação ao acessar um valor (como cálculo ou formatação), use um get com lógica.
* Propriedades derivadas: Para valores calculados com base em outros atributos, use uma propriedade com apenas get para fornecer o resultado.
  + ***Resumo:***
* Getters e Setters permitem encapsular e controlar o acesso a atributos.
* Eles fornecem um controle fino sobre como os dados são acessados e modificados, permitindo validações e lógica personalizada.
* Auto properties são uma forma simples de usar getters e setters quando você não precisa de lógica personalizada.
* Use somente get para propriedades de leitura e set privado para controle de modificação interna.
* O uso de getters e setters é crucial para garantir boas práticas de encapsulamento e manter o código mais robusto e seguro.
* ***Properties – Definir atributos:***
  + ***O que é:*** são membros de uma classe que fornecem uma forma de acessar e modificar os campos (ou atributos) de maneira controlada. Elas combinam os conceitos de **métodos de acesso** (getters e setters) com a simplicidade de acesso direto, dando a aparência de uma variável pública, mas com o controle e a lógica que um método pode oferecer.
  + ***Estrutura Básica de uma Propertie:***



* + ***Quando usar:***
* Encapsulamento de Campos: quando você deseja expor campos privados de uma classe de maneira controlada. Elas permitem que o campo seja acessado ou modificado de fora da classe, mas ainda fornecem controle interno sobre isso.
* Validação ao Atribuir Valores: Quando você precisa validar ou aplicar lógica antes de atribuir um valor a um campo, as propriedades são ideais, pois o set permite adicionar essa lógica.
* Propriedades Calculadas: Use propriedades quando o valor de um campo for derivado de outros campos e você não precisar armazenar o valor separadamente. Assim, o cálculo é feito sempre que a propriedade for acessada.
* Imutabilidade Parcial: Se você deseja permitir que um valor seja atribuído apenas no momento da inicialização do objeto e depois torná-lo imutável, use propriedades com o setprivado.
* Valores Padrão: Use propriedades quando você precisar definir valores padrão para os campos de uma classe, mas ainda permitir que eles sejam sobrescritos.
* Facilidade de Manutenção e Flexibilidade: Ao usar properties, você garante que, caso a lógica de como os dados são acessados ou modificados mude no futuro, você pode simplesmente modificar os blocos get ou set sem afetar quem já está usando a propriedade.
  + ***Resumo:***
* Sempre que quiser expor campos privados de uma classe para controle de leitura e/ou escrita, usando encapsulamento.
* Quando precisar validar os valores que estão sendo atribuídos ao campo.
* Para calcular valores dinamicamente, em vez de armazená-los como campos.
* Para garantir imutabilidade parcial, permitindo que o valor seja definido apenas na inicialização ou modificado apenas internamente.
* Quando precisar de valores padrão e ainda permitir que eles sejam sobrescritos.
  + ***Quando Evitar Properties:***
* **Quando não há necessidade de controle**: Se o valor do campo nunca precisará de validação, cálculo ou encapsulamento, você pode usar campos públicos diretamente (mas isso é raro em boas práticas de programação orientada a objetos).
* ***Auto Properties – Definir atributos:***
  + ***O que é:*** uma forma simplificada de criar propriedades em C#. Quando você usa uma auto property, o compilador automaticamente gera um campo privado "nos bastidores" para armazenar o valor. Isso economiza código e permite que você crie propriedades de maneira mais rápida, sem a necessidade de declarar explicitamente o campo privado correspondente.
  + ***Estrutura básica de uma Auto Propertie:***



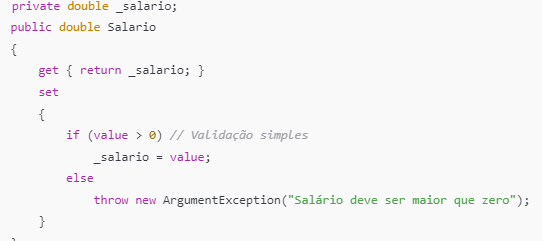
* + ***Quando usar:*** 
    - **Quando não há necessidade de lógica adicional no get ou set:** Se você só precisa de uma maneira simples de armazenar e acessar dados, sem validação ou lógica extra, as auto properties são a escolha ideal.
    - **Para manter o código simples e limpo:** Auto properties reduzem a quantidade de código boilerplate e tornam o código mais legível.
    - **Quando precisar de propriedades somente leitura ou set privado:** Auto properties com get somente ou set privado são úteis para criar imutabilidade parcial, controlando onde e como os valores podem ser modificados.
    - **Para inicializar valores com um padrão:** Quando uma propriedade deve ter um valor padrão, você pode definir isso diretamente na declaração da propriedade usando auto properties.
  + ***Benefício:*** 
    - **Simplificam o código: Menos código para escrever e manter.**
    - **Reduzem duplicidade: Você não precisa de campos privados explícitos para armazenar dados.**
    - **Encapsulamento automático: O compilador cuida dos detalhes internos.**
    - **Flexibilidade futura: Você pode adicionar lógica aos get ou set depois, se necessário, sem quebrar o código existente.**



* + ***Resumo:*** 
    - Use **auto properties** sempre que possível, especialmente em classes simples onde não há necessidade de validação ou lógica complexa.
  + ***Quando evitar Auto Properties:***
* Quando há necessidade de lógica no get ou set: Se você precisa de validação, cálculos, ou qualquer lógica adicional ao acessar ou definir o valor, você deve usar uma propriedade completa em vez de uma auto property.

Se você precisar de cálculos, validações ou outro tipo de lógica no processo de atribuição de um valor, você usaria uma property completa com um set manual (não automático).

* + - Ex:



* ***Diferença de Properties e Auto Properties:***
  + ***Properties:***
* São blocos de código que permitem o encapsulamento de um campo privado com métodos de acesso (get e set). Uma propriedade pode ter lógica personalizada para controlar a leitura ou atribuição de valores, como validação, cálculos, etc.
  + ***Auto Properties:***
* Maneira simplificada de declarar uma propriedade em C#. Elas eliminam a necessidade de declarar explicitamente um campo privado. O compilador gera automaticamente um campo privado que é usado pelos métodos get e set.
  + ***Quando usar:***
* Propriedades: Quando você precisa de controle, validação ou cálculos ao acessar ou definir valores.
* Auto Properties: Quando você só precisa de um armazenamento simples e direto de valores.
* ***Foreach():***