**TÓPICOS IMPORTANTES EM C#**

* ***Enumeração(Enum):***
  + **O que é:** é um tipo de dado especial que permite definir um conjunto de constantes nomeadas, facilitando o trabalho com valores que têm significado simbólico em vez de usar números ou strings "soltos" no código. A enum é usada para representar estados, opções ou categorias de maneira mais clara e legível.
  + **Quando usar:** As enumerações são úteis quando você tem um conjunto fixo de opções ou estados que representam algo concreto. O *enum* em C# é projetado para retornar a representação textual (nome do membro) quando usado em um contexto como Console.WriteLine ou ToString(). Se você quiser o número associado ao membro, você pode fazer um cast explícito:
  + **Vantagens:**
  1. **Legibilidade**: Em vez de usar valores numéricos ou constantes que podem não ser claros, você usa nomes simbólicos que têm um significado semântico mais forte.
  2. **Manutenção**: As enums centralizam as opções, facilitando a modificação e manutenção do código.
  3. **Segurança de tipo:** Como uma enum é um tipo de dado, você tem a garantia de que só poderá atribuir valores válidos a variáveis do tipo enum.
  + **Resumo:** Enums são uma excelente maneira de lidar com grupos de valores fixos, tornando o código mais claro, seguro e fácil de manter. Você deve usá-los sempre que tiver um conjunto definido de valores que representam algo significativo no seu sistema, como estados, opções ou categorias.
  + **Exemplo:**

enum StatusPedido

{

Pendente,

Processando,

Enviado,

Entregue,

Cancelado

}

class Pedido

{

public StatusPedido Status { get; set; }

public void AvancarStatus()

{

if (Status < StatusPedido.Cancelado)

{

Status++;

}

}

public void ExibirStatus()

{

Console.WriteLine($"O status atual do pedido é: {Status}");

}

}

class Program

{

static void Main(string[] args)

{

Pedido pedido = new Pedido();

pedido.Status = StatusPedido.Pendente;

pedido.ExibirStatus();

pedido.AvancarStatus();

pedido.ExibirStatus();

}

}

* ***Composição:*** 
  + **Conceito:** é um conceito relacionado ao design orientado a objetos, onde uma classe pode ser composta de outras classes. Basicamente, em vez de herdar comportamentos e atributos de uma classe base (como na herança), uma classe “composta” inclui instâncias de outras classes como parte de sua definição. A composição permite que você construa uma relação "tem-um" entre as classes, onde uma classe contém outra como um atributo.
  + **O que é:** A composição é quando uma classe contém uma ou mais instâncias de outras classes como membros. Esse relacionamento implica que uma classe é formada por outras, ou seja, ela "tem" objetos de outras classes. O uso de composição é uma prática que promove modularidade e reutilização de código, pois permite que as classes sejam combinadas para criar funcionalidades mais complexas.
  + **Quando usar:** composição é ideal quando uma classe "tem" outra como parte dela. Use composição:

1. Quando a relação "tem-um" faz sentido. Exemplo: um Carro tem um Motor, um Funcionario tem um Endereco.
2. Quando você quer evitar herança excessiva. A herança pode criar estruturas complexas que são difíceis de manter, enquanto a composição permite maior flexibilidade.
3. Quando você deseja criar objetos complexos a partir de objetos menores e reutilizáveis.
   * **Vantagens:**
4. **Acoplamento mais baixo**: A composição cria um acoplamento mais solto entre as classes do que a herança. Isso permite que as classes sejam alteradas sem impactar diretamente outras partes do sistema.
5. **Reusabilidade**: Diferentes classes podem compartilhar os mesmos componentes (ou instâncias de outras classes), promovendo reuso de código.
6. **Manutenção**: Facilita a manutenção, uma vez que a lógica relacionada a cada componente está isolada em classes específicas.
   * **Desvantagens:**
7. **Complexidade no gerenciamento de dependências:** Como as classes compostas podem depender de várias outras classes, isso pode aumentar a complexidade ao gerenciar dependências, especialmente quando há muitos níveis de composição.
8. **Mais código a ser gerenciado:** Comparado à herança, pode parecer que há mais código para se gerenciar, uma vez que há instâncias de outras classes envolvidas e muitas vezes construtores maiores.
   * **Diferença entre Composição e Herança:**
9. **Composição**: Relacionamento "tem-um". Exemplo: um carro tem um motor, uma empresa tem funcionários.
10. **Herança:** Relacionamento "é-um". Exemplo: um Cachorro é um Animal, um Carro é um Veículo.
    * **Resumo:** Use composição sempre que quiser construir classes a partir de outras classes, onde a relação "tem-um" seja mais adequada do que "é-um". Composição é uma abordagem flexível, utilizada em cenários onde queremos montar objetos complexos a partir de objetos menores e mais simples, promovendo modularidade e reuso de código.
    * **Exemplo:**



* ***Herança:***
  + **Conceito:** **Herança** é um princípio fundamental de programação orientada a objetos que permite que uma classe derive características (atributos e métodos) de outra. A classe "filha" ou "derivada" herda os membros da classe "pai" ou "base", tornando possível o compartilhamento de código entre classes relacionadas.
  + **O que é:** Em C#, a herança é implementada criando uma relação de "é um" (ou seja, uma relação hierárquica) entre classes. Isso significa que uma classe derivada é uma especialização de uma classe base.

Por exemplo, se temos uma classe “Conta”, podemos criar uma classe derivada “ContaPoupança" que herda todas as características e comportamentos de “Conta”, mas possibilitando adicionarmos novos atributos e métodos específicos da classe “ContaPoupança”.

* + **Quando usar:** A herança é mais útil quando há uma clara relação hierárquica entre duas classes, e a classe derivada precisa de acesso aos métodos e propriedades da classe base. Geralmente, você deve considerar usar herança se:

1. As classes compartilham atributos e comportamentos (ex.: Carro e Moto são tipos de Veiculo).
2. Deseja promover reutilização de código, evitando duplicação.
3. A relação "é um" é verdadeira (ex.: um Cachorro é um tipo de Animal).

É importante notar que o uso excessivo da herança pode tornar o código mais complexo e difícil de manter, então é preferível aplicá-la apenas quando a relação entre as classes é clara.

* + **Definições:**
    - Relação “é um”;
    - Generalização/especialização;
    - Superclasse (classe base) / Subclasse (classe derivada);
    - Herança/extensão;
    - Herança é uma associação entre classes (e não entre objetos)
    - **Palavra-chave “base”**: Permite que a classe derivada acesse membros da classe base, útil para chamadas ao construtor da classe base ou acesso a métodos.
  + **Vantagens:**
    - **Reutilização de código:** A herança permite que classes derivadas aproveitem e estendam o código da classe base, evitando duplicação e facilitando a manutenção.
    - **Organização e estrutura:** Com herança, é possível criar uma hierarquia lógica de classes, tornando o código mais compreensível e organizado.
    - **Facilidade de extensão:** A herança facilita a criação de classes especializadas que adicionam funcionalidades específicas a partir de uma classe base.
  + **Resumo:** é uma técnica de reutilização de código que permite criar hierarquias entre classes. Uma classe derivada herda os métodos e atributos de uma classe base, promovendo a reutilização e a organização do código. Ela é útil quando há uma relação hierárquica clara e quando a especialização da classe base em subclasses faz sentido. É recomendada para simplificar o desenvolvimento e a manutenção de classes relacionadas.
* ***Polimorfismo:***
  + **Conceito:** é um princípio da programação orientada a objetos que permite que métodos de classes diferentes, mas relacionadas, respondam de forma diferente à mesma chamada. Em outras palavras, polimorfismo permite o uso de uma única interface para representar múltiplos comportamentos de objetos.
  + **O que é:** Polimorfismo pode ser obtido principalmente através de duas técnicas:

1. **Polimorfismo de Sobrecarga (ou Estático):** Quando um método é sobrecarregado na mesma classe, ou seja, existem várias versões do mesmo método, cada uma com uma lista de parâmetros diferente.
2. **Polimorfismo de Substituição (ou Dinâmico):** É quando métodos em classes derivadas substituem (ou “sobrescrevem”) métodos da classe base. Isso é obtido com a palavra-chave virtual no método da classe base e override no método da classe derivada.
   * **Quando usar:** O polimorfismo é útil quando se quer tratar objetos de várias classes derivadas da mesma maneira, mas mantendo comportamentos específicos de cada um. Pode ser usado em situações como:
3. **Interfaces e classes abstratas:** Quando você deseja que classes derivadas tenham implementações específicas, mas quer tratá-las de forma uniforme.
4. Métodos que precisam ter comportamento personalizado em classes derivadas Como métodos de cálculo, exibição, entre outros.
5. **Sobrecarga de métodos:** Quando métodos precisam aceitar diferentes tipos ou números de parâmetros na mesma classe.
   * **Definições:**
     + **Polimorfismo de Sobrecarga (Estático):** Implementado com métodos com o mesmo nome e diferentes parâmetros dentro da mesma classe.
     + **Polimorfismo de Substituição (Dinâmico):** Realizado através de métodos virtual e override, permitindo que classes derivadas implementem sua própria versão de um método da classe base.
     + **Interface:** Em C#, interfaces ajudam a definir um conjunto de métodos que classes devem implementar, promovendo o polimorfismo.
     + **Classe Abstrata:** Uma classe que não pode ser instanciada diretamente e contém métodos abstract que devem serimplementados em classes derivadas, ajudando a garantir que todas as subclasses compartilhem métodos específicos.
   * **Vantagens:**
     + **Flexibilidade e extensibilidade:** Facilita a criação de código mais flexível, permitindo adicionar novos comportamentos a classes derivadas sem alterar a estrutura da classe base.
     + **Uniformidade no tratamento de objetos:** É possível usar referências de classe base para manipular objetos de classes derivadas, simplificando o código e melhorando a legibilidade.
     + **Redução de código duplicado:** Com o polimorfismo, podemos definir um comportamento geral na classe base e apenas ajustar o necessário em classes derivadas, reduzindo a duplicação de código.
   * **Resumo:** O polimorfismo em C# permite que classes derivadas redefinam o comportamento de métodos da classe base, promovendo reutilização, flexibilidade e tratamento uniforme de objetos. Ele pode ser implementado de forma estática (sobrecarga de métodos) ou dinâmica (substituição de métodos), e é especialmente útil ao trabalhar com hierarquias de classes, interfaces e classes abstratas.