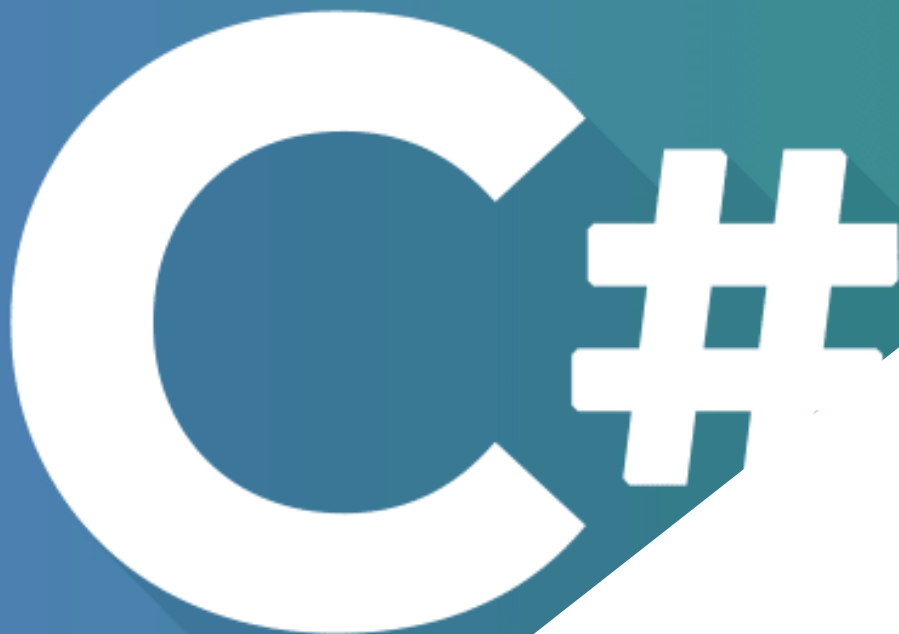


PROGRAMMING
LANGUAGE



Novidades C# 8 a 13

C# 8 – Index e Ranges

Recurso introduzido as estruturas **Index** e **Range** para acessar e manipular sequências de forma mais expressiva e eficiente.

Index permite referenciar elementos de coleções usando índices relativos, tanto a partir do início quanto do final da sequência.

- **Propriedades:**

- **IsFromEnd:** Indica se o índice é relativo ao final.
- **Value:** Retorna o valor absoluto do índice.

Range é usado para selecionar um subconjunto de elementos em coleções. Ele é definido por dois índices (start e end) e suporta sintaxe simplificada.

- **Propriedades:**

- **start** é inclusivo
- **end** é exclusivo.

C# 8 – Pattern Matching

É uma maneira de comparar valores com padrões específicos, possibilitando verificação de tipos, comparação de valores e extração de propriedades de objetos de forma simplificada.

As principais novidades são:

- **Switch Expressions**
 - Simplifica o uso do switch com uma sintaxe mais simples.
- **Property Patterns**
 - Permite verificar valores dentro de propriedades de objetos.
- **Tuple Patterns**
 - Verifica padrões baseados em tuplas.
- **Operadores de Pattern Matching**
 - **is:** Verifica se um valor corresponde a um padrão.
 - **switch:** Avalia um valor com múltiplos padrões.

C# 8 – Tipos Referência Anuláveis

Recurso que permite maior controle sobre valores null em tipos de referência, ajudando a evitar erros de referência nula. Isso é feito adicionando um contrato mais explícito para o uso de null no código.

- **Tipos referência não anuláveis:**

- string, object, etc

- **Tipos referência anuláveis:**

- string?, object?, etc (símbolo de interrogação para representar)

- **Análise de Fluxo de Nulidade:**

O compilador rastreia o fluxo do código para garantir que os tipos não anuláveis nunca contenham null.

- **Atributos para Controle de Nulidade:**

- **[NotNull]:** Garante que um parâmetro não será null após o método.
- **[MaybeNull]:** Indica que um tipo de retorno pode ser null mesmo que declarado como não anulável.

C# 9 – Init Only

Utilizado para permitir que as propriedades de uma classe ou struct sejam definidas apenas durante a inicialização do objeto, mas não possam ser alteradas depois que o objeto é criado.

- **Imutabilidade:** Garante que propriedades não possam ser alteradas após a inicialização.
- **Segurança:** Evita estados inconsistentes em objetos.
- **Flexibilidade:** Permite inicializar propriedades sem a necessidade de métodos set públicos.

Os tipos record no C# 9.0 usam propriedades init por padrão, alinhando-se ao design de objetos imutáveis.

C# 9 – Records

São um tipo especial de classe que facilita a criação de objetos com sintaxe simplificada que por sua vez trabalham com comparação pelos valores e tem como benefícios a segurança dos dados por serem imutáveis.

- **Tipos de records:**

- **Record Clássico (Referência):**

- Segue o comportamento normal de classes no que diz respeito à alocação de memória.

- **Record Struct (Tipo Valor):**

- (C# 10) Alocado na pilha e segue as regras dos tipos valor:

- **Benefícios:**

- Sintaxe simplificada;
 - Imutabilidade;
 - Comparação de igualdade;

C# 9 – With Expressions

Recurso que facilita a criação de **cópias modificadas** de objetos imutáveis. Ele é usado principalmente com **records**, mas também pode ser aplicado a tipos de referência com propriedades imutáveis (init) ou até mesmo classes tradicionais.

- **Benefícios:**

- Imutabilidade;
- Código mais limpo;
- Facilidade de uso com records;

C# 10 – Constant Interpolated Strings

Recurso que permite **interpolar** outras constantes dentro de uma string marcada com `const`. Isso permite criar strings dinâmicas baseadas em outras constantes, o que antes exigiria concatenação manual.

A interpolação é resolvida pelo compilador durante a **compilação**. Não é possível usar variáveis de instância ou valores calculados em tempo de execução.

- **Vantagens:**

- Facilidade na inicialização e Manutenção;
- Redução de Erros;
- Clareza

C# 10 – File Scoped Namespace

Oferece uma maneira mais concisa e limpa de declarar namespaces em arquivos de código, eliminando a necessidade de usar chaves {} para encapsular todo o conteúdo de um namespace.

- **Benefícios:**

- Simplicidade e clareza;
- Evita repetição de código;
- Estilo conciso;

C# 10 – GlobalUsings

Centraliza essas declarações e elas se aplicam a todos os arquivos de código do projeto, facilitando a manutenção e reduzindo a redundância, especialmente em projetos grandes.

- **Benefícios:**

- Redução de código redundante;
- Facilidade de manutenção;
- Melhor organização;

C# 11 – List Patterns

Recurso de melhoria no **Pattern Matching**, permitindo que você trabalhe com coleções, como listas ou arrays, de forma mais expressiva e concisa. Com isso, você pode usar padrões diretamente para verificar o conteúdo e a estrutura dessas coleções.

- **Benefícios:**
 - Sintaxe Clara e Intuitiva;
 - Padrões Mais Expressivos;
 - Combinação com Captura de Variáveis;
 - Trabalha Bem com Padrões Existentes;
 - Redução de Erros;
 - Melhor Performance;

C# 11 – Required Members

Recurso que garante que certas propriedades ou campos obrigatórios de uma classe ou struct sejam inicializados antes que o objeto seja usado. O compilador exige que essas propriedades sejam inicializadas antes que o objeto seja usado, reduzindo o risco de erros.

- **Benefícios:**

- Construção de objetos complexos;
- Classes com muitas propriedades que precisam de valores para serem usadas corretamente;
- Mensagem de Erro em Tempo de Compilação;

C# 11 – Static Abstract Members

Recurso que permite que interfaces definam métodos ou propriedades abstratas que são estáticos e precisam ser implementados por tipos que implementam a interface. Com esse recurso temos mais controle sobre como as classes implementadoras interagem com os membros estáticos, tornando o código mais modular, reutilizável e fácil de manter.

- **Benefícios:**

- Contratos Mais Expressivos;
- Facilidade em Criar Tipos Genéricos;
- Maior Flexibilidade
- Reusabilidade de Código;

C# 12 – Collection Expressions

Recurso que permite a criação e manipulação de coleções de maneira mais expressiva, concisa e legível, utilizando uma sintaxe baseada em **listas** (semelhante à sintaxe de inicializadores de coleções, mas com maior flexibilidade).

- **Sintaxe:**

- [...] para criar coleções.
- .. para intervalos (ranges).
- .. combinado com listas para incluir/descartar partes de coleções.
- Suporte para padrões (List Patterns) dentro das expressões.

- **Benefícios:**

- Código mais limpo;
- Evita redundância de código;
- Unificação de Sintaxe;

C# 12 – Inline Arrays

Recurso que permite a criação de arrays fixos como parte de uma estrutura (struct). Eles são usados principalmente em cenários que exigem **eficiência de memória** e **baixo overhead** (otimizações de performance). O compilador trata o array como uma **estrutura contínua de memória**, sem o overhead de referências adicionais.

- **Características:**

- **Alocação na Pilha:** São alocados na **stack** se a struct não for parte de um heap object.
- **Uso Controlado de Memória:** O tamanho é fixo e conhecido em tempo de compilação.
- **Sem GC Overhead:** Não há alocação no heap para o array, então o **Garbage Collector** não é impactado.
- **Performance:** Acesso direto e otimizado aos elementos (sem indireção).

C# 12 – Primary Constructors Non Record Types

Recurso que oferece uma sintaxe mais concisa e expressiva para inicializar objetos. Em vez de definir um método construtor tradicional, você declara um construtor primário diretamente na assinatura da classe, especificando os parâmetros e as propriedades que serão inicializadas.

- **Benefícios:**

- Código Mais Limpo;
- Consistência;
- Melhor Integração com Expressões e Métodos;

C# 13 – Params Collections

Simplifica o código e melhora o desempenho ao permitir que coleções sejam passadas diretamente para métodos que utilizam params.

É uma evolução que facilita o trabalho com coleções em cenários modernos de desenvolvimento.

- **Benefícios:**

- Flexibilidade;
- Desempenho;
- Evita conversões;

C# 13 – Scoped Lock

É um recurso que visa simplificar e tornar mais segura a utilização de locks para proteger seções críticas de código em ambientes multi-threaded. Oferece uma sintaxe mais concisa e um tratamento mais elegante de exceções, comparado ao mecanismo tradicional de lock utilizando object como argumento.

- **Características:**

- Foi introduzido para melhorar o controle de concorrência;
- Maior suporte para controle em alta concorrência;
- Maior segurança ao liberar locks e prevenir deadlocks;
- Verificação de locks em uso;