Fundamentos de Desenvolvimento com C#

Aula 14: Structs e Records

Professor: Rinaldo Ferreira Junior

E-mail: rinaldo.fjunior@prof.infnet.edu.br

- **Professor:** Rinaldo Ferreira Junior
- **Graduação:** Pós-graduado em Arquitetura de Softwares
- **Atuação:** .Net | C# | SQL | NoSQL | Engenheiro de Software
- **E-mail:** rinaldo.fjunior@prof.infnet.edu.br
- **Linkedin:** https://www.linkedin.com/in/rinaldo-ferreira-junior-787326a

Aula 14: Roteiro

- Structs
- Records

Aula 14: Memória (Heap vs Stack)

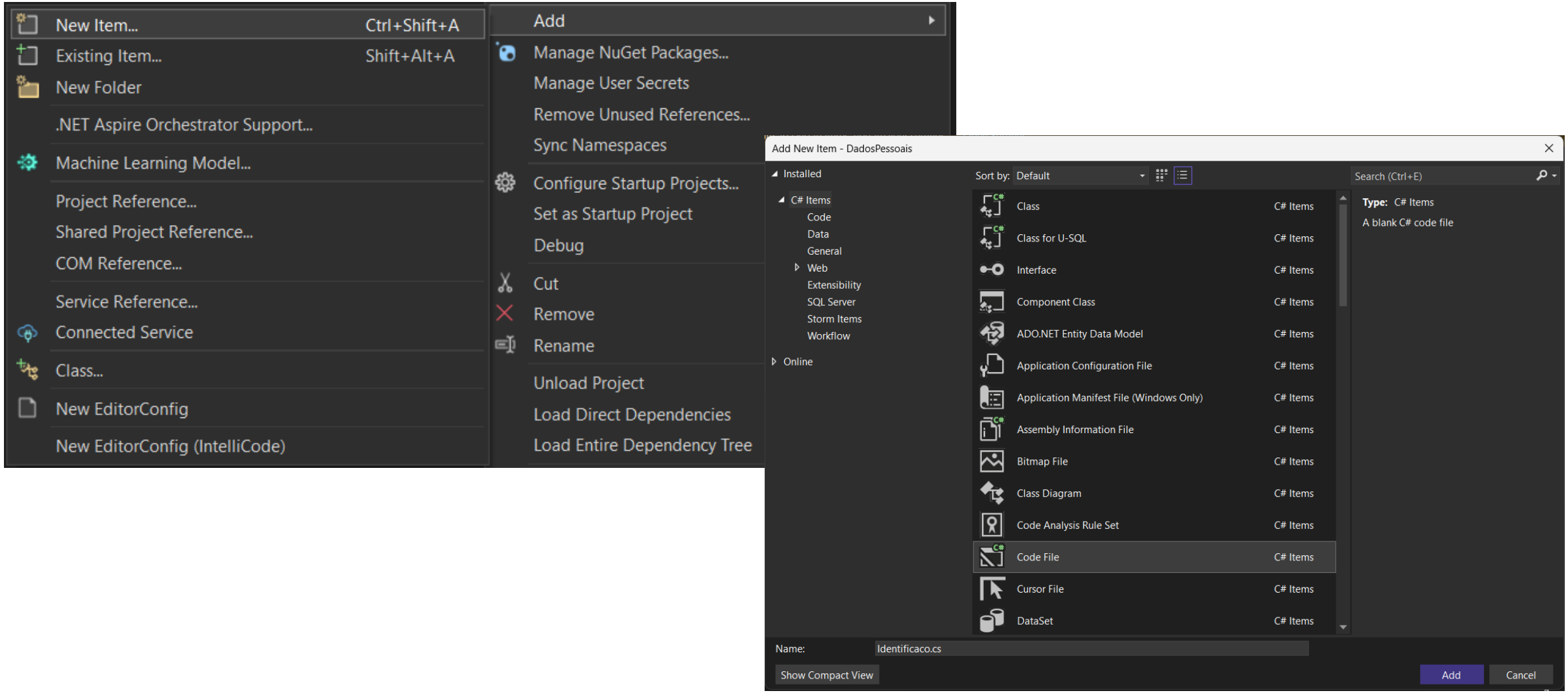
Aula 14: Structs

- São criadas como classes, mas não são Value Types. Ao contrário de classes, as structs são armazenadas no stack, enquanto classes são armazenadas no heap.
- Como structs são Value Types, uma variável contém uma cópia exata da struct.
- Em geral, structs são mais performáticas do que classes, pois o processo de alocação/desalocação de memória é mais eficiente.
- O ideal é usar structs, quando você precisa representar um conjunto de dados simples, por exemplo, coordenadas, dimensões etc.
    - Se a struct se tornar muito grande, será armazenada no heap
- Métodos de uma classe podem receber structs como parâmetros, simplificando a assinatura.
- Structs não podem ser usadas em relações de herança, mas podem implementar interfaces.

Aula 14: Structs

Instituto Infnet

- Para criar structs, clique em Adicionar – Novo Item e selecione Arquivo de Código. Não há um template específico para structs.



Aula 14: Structs

- As structs possuem membros como classes, e são invocadas como classes.

```
public struct Documentacao
{

    0 references
    public Documentacao()
    {
        this.CPF = string.Empty;
        this.RG = string.Empty;
    }

    4 references
    public string CPF { get; set; }

    3 references
    public string RG { get; set; }

    0 references
    public override string ToString()
    {
        return $"{this.RG}-{this.CPF}";
    }
}
```

```
static void Main(string[] args)
{
    Documentacao contaId = new() { RG = "212", CPF = "009" };

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaId
    };

    Console.WriteLine($"Cliente: {conta.Documentacao.RG}-{conta.Documentacao.CPF}");
    Console.Read();
}
```

Aula 14: ReadOnly Structs

Instituto Infnet

- São structs imutáveis, que não podem ter seus valores alterados, após a inicialização. Para isso, um construtor deve ser responsável por inicializar os membros da struct.

```
public readonly struct Documentacao
{

    0 references
    public Documentacao(string cpf, string rg)
    {
        this.CPF = cpf;
        this.RG = rg;
    }

    2 references
    public readonly string CPF { get; init; }

    2 references
    public readonly string RG { get; init; }

    0 references
    public override string ToString()
    {
        return $"{this.RG}-{this.CPF}";
    }
}
```

```
static void Main(string[] args)
{
    Documentacao contaId = new("212", "009");

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaId
    };

    conta.Documentacao.CPF = "333";
    Conso
    Conso
}
```

Documentacao ContaBancaria.Documentacao { get; set; }

CS8852: Init-only property or indexer 'Documentacao.CPF' can only be assigned in an object initializer, or on 'this' or 'base' in an instance constructor or an 'init' accessor.

Show potential fixes (Alt+Enter or Ctrl+.)

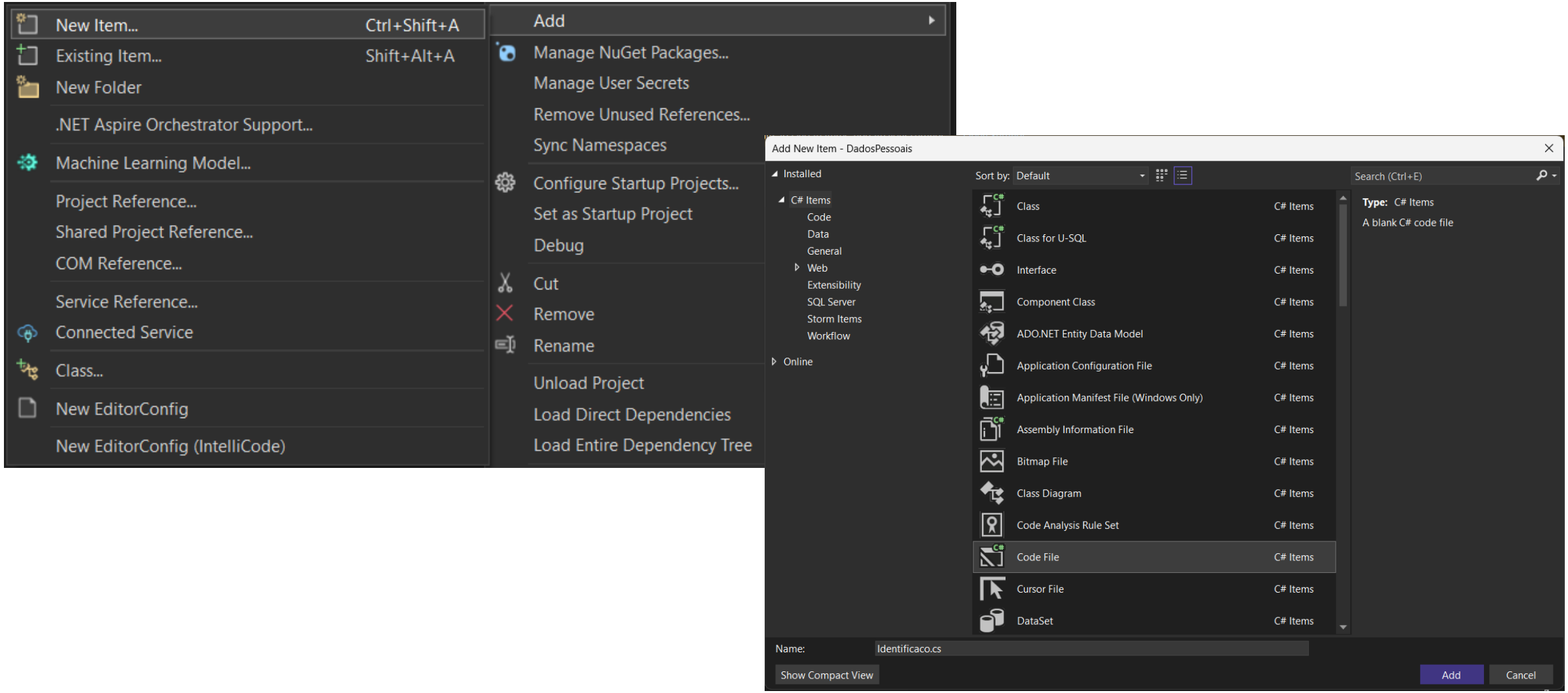
Aula 14: Records

- Disponível a partir do C# 9.0, records permitem a criação de objetos imutáveis, ou seja, objetos que não podem ser modificados após a sua criação.
  - Uma vez criado o objeto, o valor de suas propriedades não pode ser alterado.
- Se for necessário mudar um record, você gerar uma cópia dele, alterando os valores desejados.
  - A instância original permanece carregada. A cópia nada mais é, do que uma nova instância.
- Apesar de serem Reference Types, por padrão, records implementam equalidade por valor, ou seja, dois records com valores iguais em suas propriedades, são considerados iguais. Isso ocorre pela implementação da interface IEquatable.
- Os tipos record permitem deconstrução, como acontece com métodos que retornam múltiplos valores (Value Tupples).
- Por conta da imutabilidade, records são ótimos candidatos a serem usados como DTOs (Data Transfer Objects).

Aula 14: Records

Instituto Infnet

- Para criar records, clique em Adicionar – Novo Item e selecione Arquivo de Código. Não há um template específico para records.



Aula 14: Records

Instituto Infnet

- Os records possuem membros como classes, e são invocados como classes. Até por quê, são classes 😮 😎

```
public record Documentacao
{
    2 references
    public string CPF { get; set; } = string.Empty;

    2 references
    public string RG { get; set; } = string.Empty;
}
```

```
static void Main(string[] args)
{
    Documentacao contaId = new();
    contaId.RG = "212";
    contaId.CPF = "009";

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaId
    };

    Console.WriteLine($"Cliente: {conta.Documentacao.RG}-{conta.Documentacao.CPF}");
    Console.Read();
}
```

Aula 14: Records

- O grande benefício de um record é a imutabilidade. Isso é obtido através de construtores.

```
public record Documentacao(string RG, string CPF);
```

```
static void Main(string[] args)
{
    Documentacao contaId = new("212", "009");

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaId
    };

    Console.WriteLine($"Cliente: {conta.Documentacao.RG}-{conta.Documentacao.CPF}");
    Console.Read();
}
```

Aula 14: Records

- Um record pode ser parcialmente imutável.

```
public record Documentacao(string RG)
{
    2 references
    public string CPF { get; set; } = string.Empty;
}
```

```
static void Main(string[] args)
{
    Documentacao contaId = new("212");
    contaId.CPF = "009";

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaId
    };

    Console.WriteLine($"Cliente: {conta.Documentacao.RG}-{conta.Documentacao.CPF}");
    Console.Read();
}
```

Aula 14: Mutação não Destrutiva

- Um record imutável pode ser copiado e durante a cópia, seus valores podem ser alterados no novo objeto. É uma cópia, logo, não destrói o objeto original.

```
static void Main(string[] args)
{
    Documentacao contaId = new("212");
    contaId.CPF = "009";

    Documentacao contaDoc = contaId with { RG = "353" };

    ContaBancaria conta = new("CEF", 700)
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Documentacao = contaDoc
    };

    Console.WriteLine($"Cliente: {conta.Documentacao.RG}-{conta.Documentacao.CPF}");
    Console.WriteLine($"Documentação: {contaDoc.ToString()}");
    Console.Read();
}
```

Aula 14: Structs x Records

Instituto Infnet

- Structs
  - São ideais para representar valores pequenos e de representação imediata
  - Esses valores tendem a ser imutáveis.
  - Tendem a não precisar de conversões para algum Reference Type.
- Records
  - Representam algum tipo de objeto complexo (como em uma classe).
  - Deve ser imutável.
  - É usado de forma unidirecional, por exemplo, apenas para entregar dados para a interface visual.
  - Um uso amplamente divulgado, é na criação de DTOs.

Aula 14: Imutabilidade

- Elementos imutáveis apresentam alguns benefícios em sua implementação:
  - Thread Safety -> Diferentes threads não podem modificar um objeto imutável.
  - Previsibilidade -> Como seus valores não podem ser alterados, são previsíveis.
  - Consistência -> Garante valores consistentes durante a exedução do programa.
  - Error Prone -> Elimina a possibilidade de bugs, impedindo a entreda de valores inesperados.
  - Caching -> Como os dados não variam, podem facilmente ser cacheados.
- Objetos imutáveis favorecem o entendimento, por representam claramente o seu objetivo.

Aula 14: Bônus

- Para inspecionar os tipos de um programa .Net, essa ferramente é ideal. Ela faz uma leitura do IL e exibe os elementos que compoem os tipos:

- JetBrains dotPeek