Notas sobre Formatação e Estruturas de Dados

Type	Info
Topics	📚 Studies
Notebooks	Programação Profissional com Clean Code
Related Notes	Ementa do Curso

Princípios de Formatação

- Definição
  - Facilita a compreensão e colaboração da equipe
  - Reduz o tempo gasto para entender o código; diminui a probabilidade de introduzir erros

```
public class PropositoFormatacao
{
    // Exemplo Ruim: código desorganizado e difícil de
    public void ExemploRuim()
    {
        int soma = 1 + 2; Console.WriteLine(soma);
    }

    // Exemplo Bom: código bem formatado e legível
    public void ExemploBom()
    {
        int soma = 1 + 2;
        Console.WriteLine(soma);
```

```
    }
}
```

- Exemplos
  - Formatação vertifical: código organizado de forma a facilitar a leitura de cima pra baixo; inclui a organização de classes, métodos e blocos

```
public class FormatacaoVertical
{
    // Propriedade
    public int Valor { get; set; }

    // Exemplo Ruim: código não estruturado verticalmen
    public void ExemploRuim()
    {
        if (Valor > 0) { Console.WriteLine("Valor posit
    }

    // Exemplo Bom: código organizado verticalmente
    public void ExemploBom()
    {
        if (Valor > 0)
        {
            Console.WriteLine("Valor positivo");
        }
        else
        {
            Console.WriteLine("Valor não positivo");
        }
    }
}
```

  - Formatação horizontal: usar espaços a alinhamentos dentro de uma linha; pode tornar o código mais legível

```
public class FormatacaoHorizontal
{
    // Exemplo Ruim: falta de espaços e alinhamento
```

```
    public void ExemploRuim()
    {
        int resultado=1+2;Console.WriteLine(resultado);
        Calcular(1,2,3);
    }

    // Exemplo Bom: uso adequado de espaços e alinhamen
    public void ExemploBom()
    {
        // Espaços ao redor de operadores e vírgulas
        int resultado = 1 + 2;
        Console.WriteLine(resultado);

        // Espaços após vírgulas em listas de parâmetro
        Calcular(1, 2, 3);
    }

    // Método auxiliar
    private void Calcular(int a, int b, int c)
    {
        Console.WriteLine(a + b + c);
    }
}
```

- Regras da equipe: definir com a equipe sobre regras de formatação; convenções sobre indentação, espaçamento, nomenclatura

```
public class RegrasEquipe
{
    // Propriedade seguindo convenções da equipe
    public int NumeroMembros { get; set; }

    // Exemplo Ruim: não segue as regras da equipe
    public void ExemploRuim()
    {
        for (int i = 0; i < NumeroMembros; i++) { Conso
    }
```

```
    // Exemplo Bom: código consistente com as regras da
    public void ExemploBom()
    {
        for (int i = 0; i < NumeroMembros; i++)
        {
            Console.WriteLine("Membro " + i);
        }
    }
}
```

- Regras de formatação do Uncle Bob: uso consistente de indentação; linhas em branco para separar blocos de código; nomes claros e descritivos

```
public class RegrasUncleBob
{
    // Propriedade seguindo as regras de Uncle Bob
    public string Nome { get; set; }

    // Exemplo Ruim: não segue as práticas de Uncle Bob
    public void ExemploRuim()
    {
        if (Nome != "") { Console.WriteLine("Nome válid
    }

    // Exemplo Bom: seguindo as práticas de Uncle Bob
    public void ExemploBom()
    {
        // Indentação consistente e nomes claros
        if (!string.IsNullOrEmpty(Nome))
        {
            Console.WriteLine("Nome válido: " + Nome);
        }
        else
        {
            Console.WriteLine("Nome inválido");
        }
```

```
    }
}
```

- StyleCop Analyzer (pacote do .NET): aplica convenções e regras

Abstração e Encapsulamento

- Estrutura de dados são a base para o gerenciamento eficiente de dados e a implementação da lógica de negócios; utilizando as melhores práticas, será garantido a integridade dos dados, facilitar a colaboração e promover a reutilização
- Exemplos:
  - Criar interfaces públicas para esconder a implementação interna

```
public class Usuario
{
    // Exemplo Ruim: sem abstração
    public string Nome;
    public string Senha;
}

public class UsuarioMelhor
{
    // Exemplo Bom: com abstração usando propriedades a
    public string Nome { get; private set; }
    private string Senha { get; set; }

    public void DefinirNome(string value) => Nome = val

    public bool VerificarSenha(string senha) => Senha =

    public void DefinirSenha(string senha)
    {
        // Lógica de validação pode ser aplicada aqui
        Senha = senha;
    }
}
```

- Criar objetos que possuem comportamento e dados

```
public class Pedido
{
    // Exemplo Ruim: estrutura de dados
    public int Id;
    public string Produto;
    public int Quantidade;
}

public class PedidoBom
{
    // Exemplo Bom: objeto com comportamento
    private int id;
    private string produto;
    private int quantidade;

    public PedidoBom(int id, string produto, int quanti
    {
        this.id = id;
        this.produto = produto;
        this.quantidade = quantidade;
    }

    public void ExibirPedido()
    {
        Console.WriteLine($"Pedido {id}: {quantidade}x
    }
}
```

- Chamar métodos, parâmetros ou objetos criados

```
public class PedidoRuim
{
    public Cliente Cliente { get; set; }

    public void ExibirEnderecoDoCliente()
    {
```

```
        Console.WriteLine(Cliente.Endereco.Cidade.Nome)
    }
}

// Exemplo Bom: seguindo a Lei de Demeter
public class PedidoDemeter
{
    private Cliente Cliente { get; }

    public PedidoDemeter(Cliente cliente)
    {
        Cliente = cliente;
    }

    public void ExibirEnderecoDoCliente()
    {
        Console.WriteLine(Cliente.FormataEndereco());
    }
}

public class Cliente
{
    public Endereco Endereco { get; }

    public Cliente(Endereco endereco)
    {
        Endereco = endereco;
    }

    public string FormataEndereco()
    {
        return $"{Endereco.Cidade.Nome} + ";
    }
}

public class Endereco
{
    public Cidade Cidade { get; }
```

```
    public Endereco(Cidade cidade)
    {
        Cidade = cidade;
    }
}

public class Cidade
{
    public string Nome { get; }

    public Cidade(string nome)
    {
        Nome = nome;
    }
}
```

Padrões de Projeto

- Exemplos:
  - Não usar longas cadeias de código que dificultam a compreensão e manutenção

```
public class AcidenteDeTrem
{
    // Exemplo Ruim: acidente de trem
    public void ExibirEndereco(Cliente cliente)
    {
        Console.WriteLine(cliente.Endereco.Cidade.Nome)
    }
}
```

  - Não seguir características de objetos e estruturas de dados

```
public class Hibrido
{
    // Exemplo Ruim: classe híbrida
    public string Nome { get; set; }
```

```
    public string Sobrenome { get; set; }

    public string NomeCompleto()
    {
        return Nome + " " + Sobrenome;
    }

    public void AtualizarNome(string nome)
    {
        Nome = nome;
    }
}

public class HibridoMelhor
{
    // Exemplo Bom: separação de responsabilidades
    private string nome;
    private string sobrenome;

    public string NomeCompleto()
    {
        return $"{nome} {sobrenome}";
    }

    public void AtualizarNome(string nome)
    {
        this.nome = nome;
    }
}
```

- Expor apenas o necessário e esconder a implementação interna

```
public class BancoDados
{
    // Exemplo Ruim: exposição da estrutura interna
    public List<string> Usuarios { get; set; }

    public BancoDados()
```

```
    {
        Usuarios = new List<string>();
    }
}

public class BancoDadosSeguro
{
    // Exemplo Bom: esconder estrutura
    private List<string> usuarios;

    public BancoDadosSeguro()
    {
        usuarios = new List<string>();
    }

    public void AdicionarUsuario(string usuario)
    {
        usuarios.Add(usuario);
    }

    public IReadOnlyList<string> ObterUsuarios()
    {
        return usuarios.AsReadOnly();
    }
}
```

- Usar DTOs para transferir dados

```
public class UsuarioDTO
{
    // Exemplo Bom: objeto de transferência de dados
    public string Nome { get; set; }
    public string Email { get; set; }
    public DateTime DataNascimento { get; set; }
}

// Melhor ainda, utilizando Records:
public record UsuarioRDTO(string Nome, string Email, Da
```

- Não combinar dados e comportamento

```
// Exemplo Ruim: mistura de dados e comportamento
public class ProdutoRegistroAtivo
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int QuantidadeEmEstoque { get; set; }

    public void AtualizarEstoque(int quantidade)
    {
        QuantidadeEmEstoque += quantidade;
    }

    public decimal CalcularValorTotal()
    {
        return Preco * QuantidadeEmEstoque;
    }
}

// Exemplo Bom: separação de dados e comportamento
public class Produto
{
    public int Id { get; }
    public string Nome { get; }
    private decimal preco;
    private int quantidadeEmEstoque;

    public Produto(int id, string nome, decimal preco,
    {
        Id = id;
        Nome = nome;
        this.preco = preco;
        this.quantidadeEmEstoque = quantidadeEmEstoque;
    }

    public decimal Preco => preco;
```

```
    public int QuantidadeEmEstoque => quantidadeEmEstoq

    public void AtualizarEstoque(int quantidade)
    {
        quantidadeEmEstoque += quantidade;
    }

    public void AtualizarPreco(decimal novoPreco)
    {
        if (novoPreco > 0)
        {
            preco = novoPreco;
        }
    }

    public decimal CalcularValorTotal()
    {
        return preco * quantidadeEmEstoque;
    }
}
```