






Notas sobre Nomes Significativos e Comentários

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

Práticas de Nomenclatura

- Nomes devem comunicar a intenção do código; comentários fornecem informações valiosas e ajudam na compreensão (devem ser usados com inteligência)
- Importância
 - Usar nomes que revelem a intenção; por que ele existe; sem a necessidade de comentários

```
public class ExemploRevelaIntencao
{
    // Ruim: não revela a intenção
    int dp;

    // Bom: revela a intenção
    int diasDesdeInicioProjeto;
}
```

- Evitar desinformação; nomes que podem induzir ao erro

```
public class ExemploDesinformacao
{
```

```
// Ruim: confuso e desinformativo
int resultado;

// Bom: específico e claro
int numeroDePedidos;
}
```

- Práticas

- Fazer distinções significativas; nomes distintos

```
public class ExemploDistincaoSignificativa
{
    // Ruim: difícil distinguir entre as variáveis
    int valor1;
    int valor2;

    // Bom: distinções significativas
    int precoProduto;
    int quantidadeProduto;
}
```

- Usar nomes pronunciáveis; facilita a comunicação verbal sobre o código

```
public class ExemploNomesPronunciaveis
{
    // Ruim: difícil de pronunciar
    string nmProd;

    // Bom: pronunciável
    string nomeProduto;
}
```

- Usar nomes pesquisáveis; facilita a navegação e modificação

```
public class ExemploNomesPesquisaveis
{
    // Ruim: não pesquisável
```

```

    int ic;

    // Bom: pesquisável
    int idadeCliente;
}

```

- Evitar codificações e notações húngaras

```

public class ExemploCodificacoes
{
    // Ruim: notação húngara
    int iContador;

    // Bom: sem codificação
    int contador;
}

```

- Nomear interfaces e suas implementações de forma clara

```

public interface IRepositoryioCliente
{
    void AdicionarCliente(Cliente cliente);
}

public class RepositorioCliente : IRepositoryioCliente
{
    public void AdicionarCliente(Cliente cliente)
    {
        // Implementação do método
    }
}

```

- Evitar nomes que exigem o mapeamento mental

```

public class ExemploMapeamentoMental
{
    // Ruim: exige mapeamento mental
    int parcVlr;
}

```

```
// Bom: sem mapeamento mental
int valorParcial;
}
```

Nomes em Contexto

- Nomes em contexto
 - Usar nomes de classes; usar nomes substantivos que descrevam a responsabilidade

```
// Ruim: não descritivo
public class CardFluentCheck { }

// Bom: descritivo
public class ValidadorDeCartaoDeCredito { }
```

- Usar nomes de métodos; descrever exatamente a ação realizada

```
public class ExemploNomesMetodos
{
    // Ruim: não descritivo
    public void Executar() { }

    // Bom: descritivo
    public void ProcessarPagamento() { }
}
```

- Não ser fofo; evitar nomes fofos ou engraçados

```
public class ExemploNaoSejaFofo
{
    // Ruim: nome fofo
    public void VaiQueDa() { }

    // Bom: nome profissional
}
```

```
public void ProcessarTransacao() { }  
}
```

- Trocar palavra por conceito; escolher única palavra para descrever um conceito em todo o código

```
public class ExemploPalavraPorConceito  
{  
    // Ruim: inconsistência no nome  
    public void ObterCliente() { }  
    public void BuscarUsuario() { }  
    public void GetByAno() { }  
  
    // Bom: consistência no nome  
    public void ObterPedido() { }  
    public void ObterUsuario() { }  
    public void ObterTransacao() { }  
}
```

- Não fazer trocadilhos; evitar trocadilhos ou jogo de palavras

```
public class ExemploNaoFacaTrocadilhos  
{  
    // Ruim: trocadilho  
    public void DetonarPedido() { }  
  
    // Bom: direto e descritivo  
    public void CancelarPedido() { }  
}
```

- Contexto e domínio

- Usar nomes do domínio da solução para descrever a responsabilidade

```
public class CarrinhoDeCompras  
{  
    public void AdicionarItem(Item item)  
    {  
        // Lógica para adicionar item ao carrinho  
    }  
}
```

```
}  
}
```

- Usar nomes do domínio do problema

```
public class Sinistro  
{  
    public void RegistrarSinistro(DetalhesSinistro de  
    talhes)  
    {  
        // Lógica para registrar um sinistro  
    }  
}
```

- Adicionar contexto significativo; entender fora do escopo

```
// Ruim: falta de contexto  
public class Pedido  
{  
    public void Salvar()  
    {  
        // Lógica para fechar pedido  
    }  
}  
  
// Bom: contexto significativo  
public class PedidoController  
{  
    public void FecharPedido()  
    {  
        // Lógica para fechar pedido  
    }  
}
```

- Não adicionar contexto desnecessário; evitar nomes longos e confusos

```
// Ruim: contexto desnecessário  
public class SistemaDeGerenciamentoDeUsuarios
```

```

{
    public void InserirNovoUsuarioNoBancoDeDados()
    {
        // Lógica para adicionar usuário
    }
}

// Bom: contexto adequado
public class UsuarioService
{
    public void AdicionarUsuario()
    {
        // Lógica para adicionar usuário
    }
}

```

Comentários

- Comentários bons
 - Não usar comentários para compensar código ruim

```

public classCodigoRuim
{
    // Ruim: comentário tentando compensar código ruim
    int dp; // número de dias desde o início do projeto

    // Bom: código autoexplicativo
    int diasDesdeInicioProjeto; // número de dias desde o início do projeto
}

```

- Explicar no código

```

public class ExpliqueSeNoCodigo
{
    // Ruim: comentário explicando variável
}

```

```

    int ts; // tempo em segundos

    // Bom: nome de variável descritivo
    int tempoEmSegundos;
}

```

- Usar comentários legais (se necessário)

```

public class ComentariosLegais
{
    // Implementação da classe
}

```

- Fornecer informação adicional

```

public class ComentariosInformativos
{
    // Este método usa o algoritmo de Dijkstra para e
    ncontrar o caminho mais curto.
    public void EncontrarCaminhoMaisCurto() { /* ...
    */ }
}

```

- Explicar a intenção por trás do código

```

public class ExplicacaoDeIntencao
{
    // Usamos um mapa hash aqui para garantir que a p
    esquisa seja O(1).
    Dictionary<int, string> mapa = new Dictionary<in
    t, string>();
}

```

- Esclarecer partes complexas

```

public class Esclarecimento
{
    // A linha abaixo normaliza o vetor.
}

```



```
    public int vetor = Vetor.Normalizar();  
}
```

- Alertas possíveis consequências de certas ações

```
public class AvisoDeConsequencias  
{  
    // Cuidado: alterar esta variável afetará todas as  
    // instâncias da classe.  
    static int contadorGlobal;  
}
```

- Usar TODO para marcar tarefas

```
public class ComentariosTODO  
{  
    // TODO: implementar verificação de segurança  
    public void VerificarSeguranca() { /* ... */ }  
}
```

- Reforçar a compreensão (amplificação)

```
public class Amplificacao  
{  
    // O método abaixo é crítico para o desempenho da  
    // aplicação.  
    public void ProcessarDadosCriticos() { /* ... */ }  
}
```

- Comentários ruins

- Não adicionar valor (murmúrios)

```
public class Murmuriros  
{  
    // Inicializa a variável
```

```
    int x = 0;
}
```

- Ser redundante

```
public class ComentariosRedundantes
{
    // Ruim: comentário redundante
    int _contador = 0; // inicializa contador com zero

    // Bom: sem comentário redundante
    int contador = 0;
}
```

- Adicionar apenas porque "é necessário"

```
public class ComentariosObrigatorios
{
    // Método principal
    public void Main() { /* ... */ }
}
```

- Fornecer informação incorreta ou desatualizada (enganar)

```
public class ComentariosEnganosos
{
    public int CalcularSoma()
    {
        // Retorna a média (na verdade retorna a soma)
        return 1;
    }
}
```

- Usar comentários longos e detalhados (jornal)

```
public class ComentariosDeJornal
{
    // Em 24/07/2024, João alterou o método para melh
    orar a performance
    public void Calcular() { /* ... */ }
}
```

- Poluir o código (ruídos)

```
public class ComentariosDeRuido
{
    // Abaixo está o início do método
    public void Iniciar() { /* ... */ }
}
```

- Tentar explicar algo trivial

```
public class RuidoAssustador
{
    public void Main()
    {
        // Este é um loop for
        for (int i = 0; i < 10; i++) { /* ... */ }
    }
}
```

- Usar comentários ao invés de função ou variável

```
public class NaoUseComentario
{
    // Ruim: comentário explicando código
    int s = 0; // soma dos valores

    // Bom: variável descritiva
    int somaDosValores = 0;
}
```

- Marcar posição para adicionar código mais tarde

```
public class MarcadoresDePosicao
{
    // Adicionar lógica aqui
    public void Processar() { /* ... */ }
}
```

- Marcar fim do bloco de código (desnecessário)

```
public class ComentariosDeFechamentoDeChave
{
    public void Main()
    {
        // Ruim: comentário de fechamento de chave
        if (true)
        {
            // código
        } // fim do if

        // Bom: indentação basta
        if (true)
        {
            // código
        }
    }
}
```

- Descrever atribuições e assinaturas; pode ficar desatualizado ou confuso

```
public class AtribuicoesEAssinaturas
{
    public void Main()
    {
        // Ruim: comentário de atribuição
        int x = 10; // atribui 10 a x
    }
}
```

```

        // Bom: sem comentário desnecessário
        int y = 10;
    }
}

```

- Comentar código (verificar se pode ser removido)

```

public classCodigoComentado
{
    // Ruim: código comentado
    // int y = 20;

    // Bom: código limpo
    int z = 30;
}

```

- Usar comentários HTML

```

public classComentariosHTML
{
    // Ruim: comentário HTML
    //<!-- Este é um comentário HTML -->

    // Bom: comentário de código
    // Este é um comentário de código
}

```

- Fornecer informação não local

```

public classInformacaoNaoLocal
{
    // Verificar se o usuário está autenticado
    // (este comentário deve estar perto do código de
    autenticação)
    public void Autenticar() { /* ... */ }
}

```

- Fornecer informação demais

```
public class InformacaoDemais
{
    // Este método faz isso, isso e aquilo e foi criado para
    // resolver tal problema porque tivemos uma reunião em 01/01/2024...
    public void Complexo() { /* ... */ }
}
```

- Fornecer informação não óbvia

```
public class ConexaoNaoObvia
{
    // A variável abaixo é usada no método X (certifique-se de que isso é óbvio no código)
    int variavelImportante;
}
```

- Usar cabeçalho de função de maneira confusa

```
public class CabecalhosDeFuncao
{
    // Calcula a soma de dois números.
    public int Somar(int a, int b)
    {
        return a + b;
    }
}
```