

# **Código Limpo: Princípios e Práticas**

Boas Práticas de Programação - BPP 2025.2

---

Prof. Fernando Marques Filho

29 de agosto de 2025

Universidade Federal do Rio Grande do Norte

# Agenda

# Introdução ao Código Limpo

---

# O que é Código Limpo?

## Código Limpo

É um código  
fácil de entender e fácil de alterar

*"Qualquer tolo consegue escrever código que um computador entende.  
Bons programadores escrevem código que humanos podem entender."*

— Martin Fowler

# Origem e Motivação

- **Criado por:** Robert C. Martin (Uncle Bob)
- **Motivação:** Combater o débito técnico
- **Objetivo:** Software sustentável e de qualidade

## Débito Técnico

Custo implícito de uma implementação pensada apenas no agora, em vez de usar uma abordagem de melhor qualidade.



# Nomes Significativos

---

# Nomes Significativos

## ✗ Ruim

```
1 // Nomes que n o revelam
   prop sito
2 int d; // dias decorridos
3 String nm = "Jo o";
4 List<String> lst = new ArrayList
   <>();
5
6 // Dif cil de entender o contexto
7 double x = calculateSalary(h, r);
```

## ✓ Bom

```
1 // Nomes que revelam prop sito
   claramente
2 int daysSinceLastLogin;
3 String customerName = "Jo o";
4 List<String> approvedUsers = new
   ArrayList<>();
5
6 // Fica claro o prop sito de cada
   vari vel
7 // F cil de entender e manter
8 double monthlySalary =
   calculateMonthlySalary(
   hoursWorked, hourlyRate);
```

# Nomes Pronunciáveis e Buscáveis

## ❌ Evite

```
1 // Nomes difíceis de pronunciar e
   buscar
2 String x1zqp = "data";
3 int grmblwskx = 42;
4 List<User> usrLstMngr = new
   ArrayList<>();
5
6 class XyzManager {
7     // Difícil de encontrar no
   código
8     // Difícil de pronunciar em
   reuniões
9     void prcssXyzData() { }
10 }
```

## ✅ Prefira

```
1 // Nomes pronunciáveis e
   buscáveis
2 String applicationData = "data";
3 int maxRetryAttempts = 42;
4 List<User> userManager = new
   ArrayList<>();
5
6 class AccountManager {
7     // Fácil de encontrar com Ctrl
   +F
8     // Fácil de pronunciar e
   discutir
9     void processAccountData() { }
10 }
```



# Classes e Métodos: Nomenclatura

## ✗ Classes Genéricas

```
1 // Nomes genéricos que não  
   revelam propósito  
2 class Manager { }  
3 class Data { }  
4 class Info { }  
5 class Processor { }  
6 class Handler { }
```

## ✓ Classes Específicas

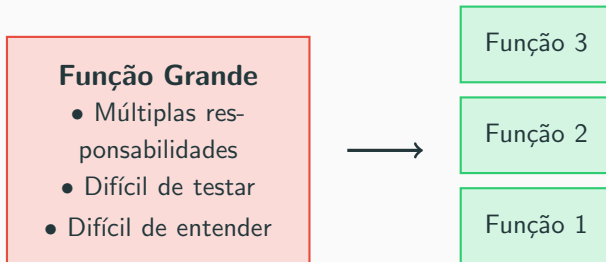
```
1 // Nomes específicos que revelam  
   propósito  
2 class UserManager { }  
3 class CustomerData { }  
4 class OrderInformation { }  
5 class PaymentProcessor { }  
6 class EmailNotificationHandler { }
```

- **Classes:** Substantivos ou frases nominais
- **Métodos:** Verbos ou frases verbais
- **Evite:** Manager, Processor, Data, Info

# Funções

---

## Funções: Pequenas e Focadas



### Regra

Uma função deve fazer uma coisa, fazê-la bem e fazer apenas ela.

## Exemplo: Refatoração de Função

### ❌ Função com Múltiplas Responsabilidades

```
1 # Função com múltiplas responsabilidades
2 def processar_tarefas(tarefas):
3     for tarefa in tarefas:
4         if tarefa.status == "pendente":
5             print("Descrição:", tarefa.descricao)
6             enviar_email(tarefa.usuario.email)
7             tarefa.status = "processado"
8             salvar_banco(tarefa)
```

### ✅ Funções Especializadas

```
1 # Funções especializadas e focadas
2
3 def filtrar_tarefas_pendentes(tarefas):
4     return [t for t in tarefas if t.status == "pendente"]
5
6 def notificar_usuario(tarefa):
```

# Argumentos de Função

0

Ideal

1

Bom

2

Aceitável

3

Evitar

3+

Refatorar

## ✗ Muitos Parâmetros

```
1 // Muitos parâmetros - difícil de
  usar
2 public void criarUsuario(String
  nome, String email,
3                               String
  telefone, String endereco,
4                               int idade,
```

## ✓ Objeto de Parâmetro

```
1 // Objeto de parâmetro - mais
  claro e flexível
2 public class DadosUsuario {
3     public String nome;
4     public String email;
5     public String telefone;
6     public String endereco;
7     public int idade;
8     public boolean ativo;
9     public String departamento;
```

# Efeitos Colaterais em Funções

## Problema

Funções que fazem mais do que prometem em seu nome causam efeitos colaterais inesperados.

### ✗ Com Efeito Colateral

```
1 // Função com efeito colateral inesperado
2 public boolean validarSenha(String
   senha) {
3     if (senha.length() > 8) {
4         // Efeito colateral: inicia
           sessão
5         iniciarSessao();
6         return true;
7     }
8     return false;
```

### ✓ Sem Efeito Colateral

```
1 // Separa o clara de
   responsabilidades
2 public boolean validarSenha(String
   senha) {
3     return senha.length() > 8;
4 }
5
6 public void fazerLogin(String senha
   ) {
7     if (validarSenha(senha)) {
8         iniciarSessao();
```

## Comentários

---

*"A necessidade de comentários muitas vezes indica  
que o código não está claro o suficiente"*  
— Uncle Bob

### ✓ Bons Comentários

- Explicação de intenções
- Esclarecimentos
- Avisos de consequências
- TODOs

### ✗ Maus Comentários

- Murmúrios
- Redundantes
- Enganosos
- Código comentado

### **i** Regra

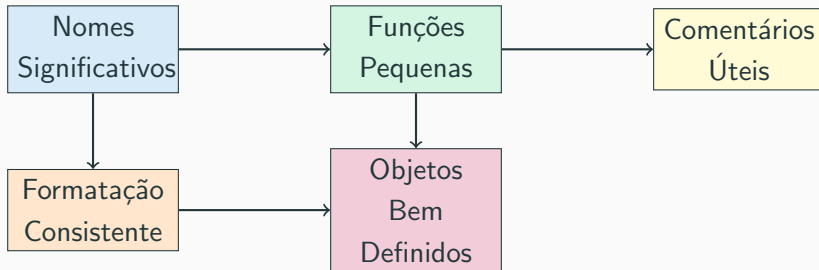
Escreva código autoexplicativo primeiro. Use comentários apenas quando necessário.



## Conclusão

---

## Resumo dos Princípios



### Lembre-se

Código limpo não é escrito de uma vez. É refinado continuamente.

## Próximos Passos

1. **Pratique:** Aplique esses princípios no seu código diário
2. **Refatore:** Melhore código existente gradualmente
3. **Code Review:** Use esses critérios para avaliar código
4. **Ferramentas:** Utilize analisadores estáticos (próxima aula)

### Para a Próxima Aula

- Leitura: Capítulo sobre Code Smells
- Exercício: Identificar code smells em projeto pessoal
- Ferramenta: Instalar SonarQube/PMD

Dúvidas?

## Referências

- Martin, Robert C. **Clean Code: A Handbook of Agile Software Craftsmanship**. Prentice Hall, 2008.
- Fowler, Martin. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, 2019.
- Catálogo de Code Smells: <https://luzkan.github.io/smells/>
- Repositório do curso: <https://github.com/fmarquesfilho/bpp-2025-2>

Obrigado pela atenção!

`fernando@dimap.ufrn.br`