

# Capítulo 9 – Refactoring

---

## 1. Introdução

---

Refactoring refere-se a **modificações no código** que melhoram sua **legibilidade, organização e manutenibilidade**, sem alterar o funcionamento do sistema.

A necessidade de refatoração surge devido à **evolução natural do software**. Segundo as **Leis de Lehman**, sistemas de software envelhecem com o tempo, tornando-se mais complexos e difíceis de manter. O **refactoring** combate esse envelhecimento ao melhorar a estrutura do código sem modificar sua funcionalidade.

O conceito foi formalizado por **Martin Fowler** em 2000, com um catálogo de técnicas que ajudam a melhorar código existente. Ele enfatiza que refactoring deve ser feito **continuamente**, para evitar acúmulo de "dívida técnica".

## 2. Principais Técnicas de Refactoring

---

### 2.1 Extração de Método

Separa trechos de código repetitivos ou longos em métodos menores e mais legíveis.

Exemplo Antes:

```
void calcularSalario() {  
    double imposto = salario * 0.2;  
    double bonus = salario * 0.1;  
    salario = salario - imposto + bonus;  
}
```

Após Refatoração:

```
void calcularSalario() {  
    double imposto = calcularImposto();  
    double bonus = calcularBonus();  
    salario = salario - imposto + bonus;  
}  
  
double calcularImposto() { return salario * 0.2; }  
double calcularBonus() { return salario * 0.1; }
```

### 2.2 Inline de Método

Remove métodos desnecessários e insere seu código diretamente no chamador.

Exemplo Antes:

```
double calcularImposto() { return salario * 0.2; }  
void calcularSalario() { salario = salario - calcularImposto(); }
```

Após Refatoração:

```
void calcularSalario() { salario = salario - (salario * 0.2); }
```

## 2.3 Movimentação de Método

Transfere um método para outra classe quando ele acessa mais dados de outra classe do que da própria.

Exemplo Antes:

```
class Cliente {  
    Endereco endereco;  
    String getCep() { return endereco.cep; }  
}
```

Após Refatoração:

```
class Endereco {  
    String cep;  
    String getCep() { return cep; }  
}  
class Cliente {  
    Endereco endereco;  
}
```

## 2.4 Extração de Classe

Quando uma classe tem muitas responsabilidades, uma parte dela pode ser movida para uma nova classe.

Exemplo Antes:

```
class Cliente {  
    String nome;  
    String telefone;  
    String email;  
}
```

Após Refatoração:

```
class Contato {  
    String telefone;  
    String email;  
}  
class Cliente {  
    String nome;  
    Contato contato;  
}
```

## 2.5 Renomeação de Método ou Variável

Melhora a clareza do código alterando nomes pouco intuitivos.

Exemplo Antes:

```
void c() { /* código */ }
```

Após Refatoração:

```
void calcularTotal() { /* código */ }
```

## 3. Code Smells (Indícios de Código Ruim)

---

Code smells são sinais de que um código precisa ser refatorado. Alguns exemplos incluem:

- **Código Duplicado:** Pode ser eliminado com **Extração de Método**.
- **Métodos Longos:** Melhorados com **Extração de Método** para dividir em partes menores.
- **Classes Grandes:** Podem ser divididas usando **Extração de Classe**.
- **Feature Envy:** O método pode ser movido para outra classe com **Movimentação de Método**.
- **Variáveis Globais:** Devem ser encapsuladas em classes específicas.

## 4. Estratégias para Aplicação de Refactoring

---

### 4.1 Refactoring Oportunista

- Realizado durante a implementação de novas funcionalidades.
- Pequenos ajustes no código para evitar o acúmulo de problemas.

### 4.2 Refactoring Planejado

- Realizado como uma **tarefa separada**, em situações onde grandes melhorias na estrutura do sistema são necessárias.

Recomenda-se combinar ambos os tipos para manter o código saudável!

## 5. Ferramentas para Refactoring

---

A maioria das IDEs modernas (Eclipse, IntelliJ, Visual Studio) oferecem suporte a refactorings automáticos, como:

- Renomeação de métodos e variáveis.
- Extração de métodos e classes.
- Movimentação de métodos entre classes.
- Detecção de código duplicado.

Essas ferramentas ajudam a evitar erros e garantem que a refatoração *preserve* o comportamento original do sistema.