



INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Linguagem de Programação II

Introdução a Refatoração (Refactoring);



Roteiro

- Introdução a Refatoração (Refactoring);
 - O que é Refatoração (Refactoring)?
 - Por que Refatorar?
 - Princípios de Refatoração (Refactoring);
 - Técnicas de Refatoração (Refactoring);
 - Conclusão.



O que é Refatoração (Refactoring) ?

É o processo de reestruturar o código existente sem alterar o seu comportamento externo.

O objetivo é melhorar a legibilidade, a manutenibilidade e a eficiência do código.



Por que Refatorar ?

Legibilidade e Clareza: Código refatorado é mais fácil de ler e entender. Isso ajuda na colaboração e na manutenção a longo prazo.

Redução de Duplicação: Refactoring pode eliminar código duplicado, tornando o código mais conciso e menos propenso a erros.



Por que Refatorar ?

Facilita a Manutenção: Código bem-refatorado é mais fácil de manter e corrigir bugs.

Melhora a Eficiência: O código refatorado muitas vezes é mais eficiente em termos de desempenho.



Por que Refatorar ?

Prepara para Novas Funcionalidades: Um código bem-refatorado é mais fácil de estender com novos recursos.



Princípios de Refatoração (Refactoring)

Mantenha Testes Automatizados: Antes de refatorar, certifique-se de ter uma boa cobertura de testes automatizados. Isso ajuda a garantir que as mudanças não quebrem o código.



Princípios de Refatoração (Refactoring)

Pequenos Passos: Refatore em pequenos passos. Cada passo deve ser pequeno o suficiente para ser feito em alguns minutos.



Princípios de Refatoração (Refactoring)

Mantenha o Código Funcionando: A cada passo, o código deve continuar funcionando como antes.

Nomes Significativos: Use nomes descritivos para variáveis, métodos e classes. Isso torna o código mais legível.



Princípios de Refatoração (Refactoring)

Simplicidade: Mantenha o código o mais simples possível. Evite complexidades desnecessárias.



Técnicas de Refatoração (Refactoring)

Extrair Método

Esta técnica envolve a criação de um novo método para um bloco de código que pode ser isolado e reutilizado.



Técnicas de Refatoração (Refactoring)

Antes:

```
public void calcularSalario() {  
    //Código para calcular o salário:  
    //--Código para calcular os impostos  
    //--Código para calcular os benefícios  
  
    System.out.println("Salário calculado.");  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public void calcularSalario() {  
    calcularImpostos();  
    calcularBeneficios();  
  
    System.out.println("Salário calculado.");  
}  
  
public void calcularImpostos() {  
    //Código para calcular os impostos  
}  
  
public void calcularBeneficios() {  
    //Código para calcular os benefícios  
}
```



Técnicas de Refatoração (Refactoring)

Renomear Variáveis (Atributos) e Métodos

Use nomes descritivos e claros para facilitar a compreensão do código.



Técnicas de Refatoração (Refactoring)

Antes:

```
public class Circulo {  
  
    double r;  
  
    public double calcArea() {  
        return Math.PI * r * r;  
    }  
  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public class Circulo {  
  
    double raio;  
  
    public double calcularArea() {  
        return Math.PI * raio * raio;  
    }  
  
}
```



Técnicas de Refatoração (Refactoring)

Eliminar Código Desnecessário (Morto)

Remova código que não está sendo utilizado para manter o código limpo e mais fácil de entender.



Técnicas de Refatoração (Refactoring)

Antes:

```
public void adicionarProduto(int quantidade) {  
    int totalItens = 0;  
    int quantidadeFinal = 0;  
  
    totalItens += quantidade;  
  
    System.out.println("Total: " + totalItens);  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public void adicionarProduto(int quantidade) {  
    int totalItens = 0;  
  
    totalItens += quantidade;  
  
    System.out.println("Total: " + totalItens);  
}
```



Técnicas de Refatoração (Refactoring)

Dividir Classes ou Métodos Grandes

Se uma classe ou método está fazendo muitas coisas, considere dividi-los em partes menores e mais específicas.



Técnicas de Refatoração (Refactoring)

Antes:

```
public class Exemplo {  
    public void metodoGrande () {  
        //Parte 1 do código  
        //Parte 2 do código  
        //Parte 3 do código  
    }  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public class Exemplo {  
    public void parte1() {  
        //Parte 1 do código  
    }  
  
    public void parte2() {  
        //Parte 2 do código  
    }  
  
    public void parte3() {  
        //Parte 3 do código  
    }  
}
```



Técnicas de Refatoração (Refactoring)

Mover Métodos ou Atributos

Mova métodos ou atributos para a classe onde faz mais sentido em termos de responsabilidade.



Técnicas de Refatoração (Refactoring)

Antes:

```
public class Pessoa {  
    private String nome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
  
    public void exibir() {  
        System.out.println("Nome: " + nome);  
    }  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public class Pessoa {  
    private String nome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public class InterfaceConsole {  
    public static void exibir(Pessoa pessoa) {  
        System.out.println("Nome: " + pessoa.getNome());  
    }  
}
```



Técnicas de Refatoração (Refactoring)

Consolidar Condições

Se houver múltiplas condições semelhantes, considere consolidá-las para tornar o código mais conciso.



Técnicas de Refatoração (Refactoring)

Antes:

```
public boolean elegivel(int idade, int experiencia, boolean diploma) {  
    if (idade >= 18) {  
        if (experiencia >= 2) {  
            if (diploma) {  
                return true;  
            }  
        }  
    }  
    return false;  
}
```



Técnicas de Refatoração (Refactoring)

Depois:

```
public boolean elegivel(int idade, int experiencia, boolean diploma) {  
    return (idade >= 18 && experiencia >= 2 && diploma);  
}
```



Técnicas de Refatoração (Refactoring)

Estas são apenas algumas das técnicas comuns de refactoring.

Cada uma delas tem um propósito específico e pode ser aplicada em diferentes situações.



Conclusão

Refatoração (Refactoring) é uma prática essencial na engenharia de software que ajuda a manter o código limpo, legível e fácil de manter.

Ao aplicar técnicas de refactoring, você pode melhorar a qualidade do código e a eficiência do desenvolvimento.



Exercícios – Pesquisa e Resposta

Exercícios:

- 1) O que é Refatoração (Refactoring)?
- 2) Por que é adequado Refatorar um Software?
- 3) Quais os Princípios da Refatoração (Refactoring)?
- 4) Cite Técnicas de Refatoração (Refactoring) além das estudadas.
- 5) Quando (no código) é Adequado Atilizar a Técnica de Refatoração (Refactoring) de Extração de Método?



Referências

- VALENTE, Marco Tulio. Engenharia de software moderna. Princípios e Práticas para Desenvolvimento de Software com Produtividade, v. 1, 2020.
- <https://engsoftmoderna.info/cap9.html>



Obrigado!

- Canais de Comunicação;
- Horário de Atendimento.

