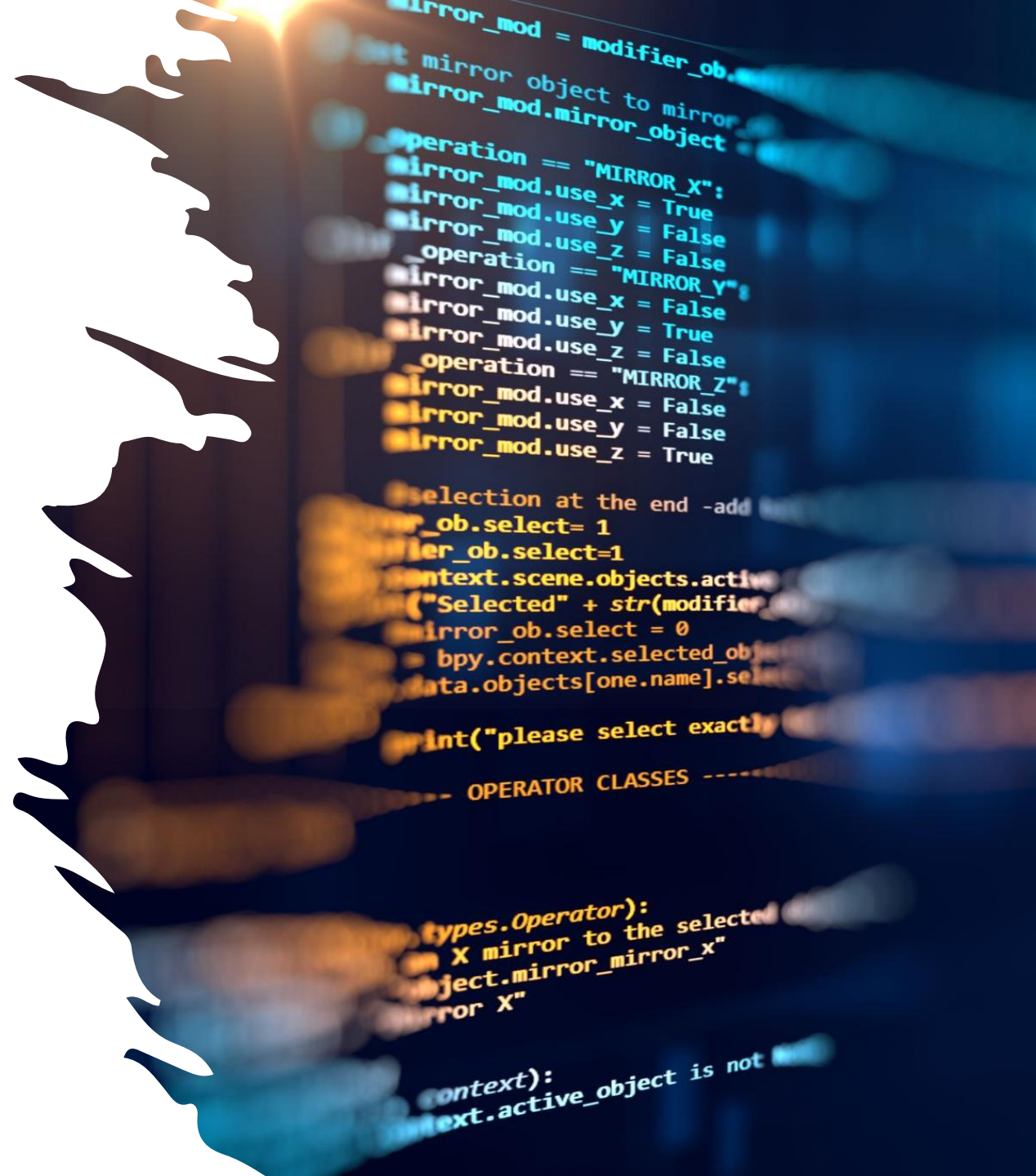


# Boas práticas de programação

## Especialização em desenvolvimento Web – Turma 6

PROFESSOR: DR. RODRIGO  
DA CRUZ FUJIOKA

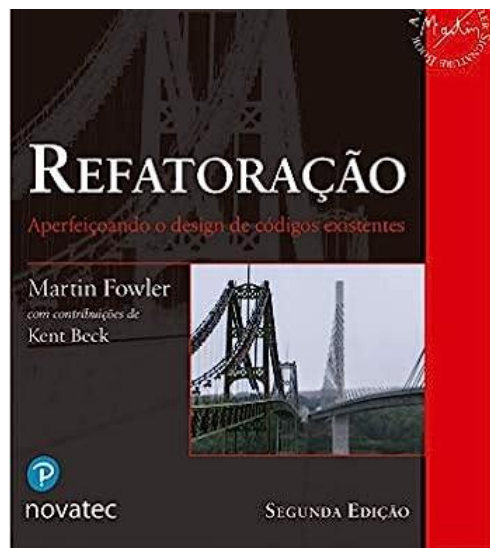


```
mirror_mod = modifier_ob.  
Set mirror object to mirror.  
mirror_mod.mirror_object =  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True
```

```
selection at the end -add  
obj.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly
```

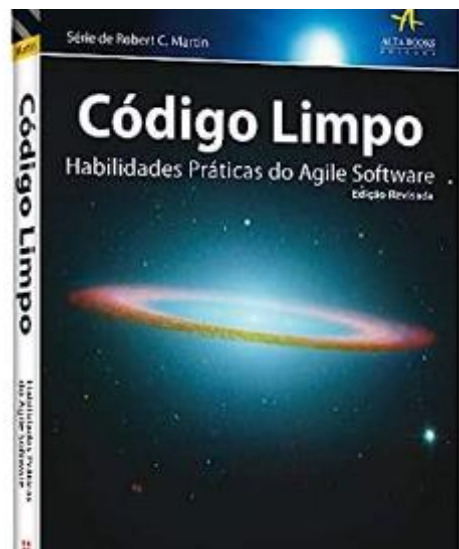
```
-- OPERATOR CLASSES --  
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"
```

```
context):  
context.active_object is not
```



# Recomendações de Leitura

(REFATORAÇÃO DE CÓDIGO E  
SOLID)



```
List<GráficoEntregaArmaDTO> entregasArmas = new ArrayList<>();
LinkedHashMap<String, Long> mapaSerieArmas = new LinkedHashMap<>();

montarLista0Ate12Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista13Ate18Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista19Ate25Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista26Ate30Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista31Ate35Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista36Ate40Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista41Ate45Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista46Ate50Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista51Ate55Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista56Ate60Anos(entregadorExportDTOList, parametrosGráficoDTO);
montarLista60Mais(entregadorExportDTOList, parametrosGráficoDTO);
```

```
List<Integer> ids = Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
```

```
if(entregadorFilterDTO.getFaixaEtaria() == null) {
    for(Integer id : ids) {
        calcularSerieDeAcordoComAFaixa(entregadorFilterDTO, entregasArmas, mapaSerieArmas, id);
    }
} else {
    for (Integer id : entregadorFilterDTO.getFaixaEtaria()) {
        calcularSerieDeAcordoComAFaixa(entregadorFilterDTO, entregasArmas, mapaSerieArmas, id);
    }
}
```

```
return entregasArmas;
```

```
2 usages
private void calcularSerieDeAcordoComAFaixa(EntregadorFilterDTO entregadorFilterDTO, List<GráficoEntregaArmaDTO> entregasArmas,
LinkedHashMap<String, Long> mapaSerieArmas, Integer id) {
```

```
if(id == 0) {
    calcularSerieArmaFaixa0Ate12Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 1) {
    calcularSerieArmaFaixa13Ate18Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 2) {
    calcularSerieArmaFaixa19Ate25Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 3) {
    calcularSerieArmaFaixa26Ate30Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 4) {
    calcularSerieArmaFaixa31Ate35Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 5) {
    calcularSerieArmaFaixa36Ate40Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 6) {
    calcularSerieArmaFaixa41Ate45Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 7) {
    calcularSerieArmaFaixa46Ate50Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 8) {
    calcularSerieArmaFaixa51Ate55Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 9) {
    calcularSerieArmaFaixa56Ate60Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 10) {
    calcularSerieArmaFaixa60Mais(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}
```

```
if(id == 0) {
    calcularSerieArmaFaixa0Ate12Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 1) {
    calcularSerieArmaFaixa13Ate18Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 2) {
    calcularSerieArmaFaixa19Ate25Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 3) {
    calcularSerieArmaFaixa26Ate30Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 4) {
    calcularSerieArmaFaixa31Ate35Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 5) {
    calcularSerieArmaFaixa36Ate40Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 6) {
    calcularSerieArmaFaixa41Ate45Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 7) {
    calcularSerieArmaFaixa46Ate50Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 8) {
    calcularSerieArmaFaixa51Ate55Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 9) {
    calcularSerieArmaFaixa56Ate60Anos(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}

if(id == 10) {
    calcularSerieArmaFaixa60Mais(entregasArmas, mapaSerieArmas, parametrosGráficoDTO, entregadorFilterDTO.getSexo());
}
```

```
1 usage
private void calcularSerieArmaFaixa0Ate12Anos(List<GráficoEntregaArmaDTO> entregasArmas, LinkedHashMap<String, Long> mapaSerieArmas, Integer id,
ParametroGráficoDTO parametrosGráficoDTO, String sexo) {
```

```
var armasFemininas = 0L;
var armasMascullinas = 0L;
var munitoesFemininas = 0L;
var munitoesMascullinas = 0L;

if(sexo == null) {
    armasMascullinas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalArmasMascullinas());
    armasFemininas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalArmasFemininas());
    munitoesMascullinas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalMunitoesMascullinas());
    munitoesFemininas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalMunitoesFemininas());
}

if(sexo != null && sexo.equals("M")) {
    armasMascullinas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalArmasMascullinas());
    munitoesMascullinas = retornarValorFaixaDeAcordoComIdadeESexo(parametrosGráficoDTO.getTotalMunitoesMascullinas());
}
```

```
1 usage
private void montarLista31Ate35Anos(List<EntregadorExportDTO> entregadorExportDTOList, ParametroGráficoDTO parametrosGráficoDTO, String sexo) {

    var totalArmasMascullinoFaixa31Ate35 = 0L;
    var totalArmasFemininoFaixa31Ate35 = 0L;
    var totalMunitoesMascullinoFaixa31Ate35 = 0L;
    var totalMunitoesFemininoFaixa31Ate35 = 0L;

    for(EntregadorExportDTO entregadorExportDTO : entregadorExportDTOList) {
        if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_MASCULLINO) &&
            && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_31_35_ANOS)) {
            totalArmasMascullinoFaixa31Ate35 = totalArmasMascullinoFaixa31Ate35 + entregadorExportDTO.getTotalArmasMascullinoFaixa31Ate35();
        } else if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_FEMININO) &&
            && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_31_35_ANOS)) {
            totalArmasFemininoFaixa31Ate35 = totalArmasFemininoFaixa31Ate35 + entregadorExportDTO.getTotalArmasFemininoFaixa31Ate35();
        }

        if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_MASCULLINO) &&
            && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_31_35_ANOS)) {
            totalMunitoesMascullinoFaixa31Ate35 = totalMunitoesMascullinoFaixa31Ate35 + entregadorExportDTO.getTotalMunitoesMascullinoFaixa31Ate35();
        } else if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_FEMININO) &&
            && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_31_35_ANOS)) {
            totalMunitoesFemininoFaixa31Ate35 = totalMunitoesFemininoFaixa31Ate35 + entregadorExportDTO.getTotalMunitoesFemininoFaixa31Ate35();
        }

        parametrosGráficoDTO.setTotalMunitoesFemininoFaixa31Ate35(totalMunitoesFemininoFaixa31Ate35);
        parametrosGráficoDTO.setTotalMunitoesMascullinoFaixa31Ate35(totalMunitoesMascullinoFaixa31Ate35);
        parametrosGráficoDTO.setTotalArmasFemininoFaixa31Ate35(totalArmasFemininoFaixa31Ate35);
        parametrosGráficoDTO.setTotalArmasMascullinoFaixa31Ate35(totalArmasMascullinoFaixa31Ate35);

        if(totalArmasMascullinoFaixa31Ate35 > 0 || totalArmasFemininoFaixa31Ate35 > 0 ||
            totalMunitoesFemininoFaixa31Ate35 > 0 || totalMunitoesMascullinoFaixa31Ate35 > 0) {
            List<String> faixas = new ArrayList<>();
            faixas = montarSerieFaixasPorIdade(parametrosGráficoDTO, faixas);
            parametrosGráficoDTO.setFaixasSalvas(faixas);
        } else {
            parametrosGráficoDTO.setFaixa31Ate35(null);
            List<String> faixas = new ArrayList<>();
            faixas = montarSerieFaixasPorIdade(parametrosGráficoDTO, faixas);
            parametrosGráficoDTO.setFaixasSalvas(faixas);
        }
    }
}
```

```
1 usage
private void montarLista36Ate40Anos(List<EntregadorExportDTO> entregadorExportDTOList, ParametroGráficoDTO parametrosGráficoDTO, String sexo) {
```

```
var totalArmasMascullinoFaixa36Ate40 = 0L;
var totalArmasFemininoFaixa36Ate40 = 0L;
var totalMunitoesMascullinoFaixa36Ate40 = 0L;
var totalMunitoesFemininoFaixa36Ate40 = 0L;

for(EntregadorExportDTO entregadorExportDTO : entregadorExportDTOList) {
    if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_MASCULLINO) &&
        && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_36_40_ANOS)) {
        totalArmasMascullinoFaixa36Ate40 = totalArmasMascullinoFaixa36Ate40 + entregadorExportDTO.getTotalArmasMascullinoFaixa36Ate40();
    } else if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_FEMININO) &&
        && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_36_40_ANOS)) {
        totalArmasFemininoFaixa36Ate40 = totalArmasFemininoFaixa36Ate40 + entregadorExportDTO.getTotalArmasFemininoFaixa36Ate40();
    }

    if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_MASCULLINO) &&
        && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_36_40_ANOS)) {
        totalMunitoesMascullinoFaixa36Ate40 = totalMunitoesMascullinoFaixa36Ate40 + entregadorExportDTO.getTotalMunitoesMascullinoFaixa36Ate40();
    } else if (entregadorExportDTO.getSexo().equals(ParametroGráficoDTO.SEXO_FEMININO) &&
        && entregadorExportDTO.getFaixaEtaria().equals(ParametroGráficoDTO.FAIXA_36_40_ANOS)) {
        totalMunitoesFemininoFaixa36Ate40 = totalMunitoesFemininoFaixa36Ate40 + entregadorExportDTO.getTotalMunitoesFemininoFaixa36Ate40();
    }

    parametrosGráficoDTO.setTotalMunitoesFemininoFaixa36Ate40(totalMunitoesFemininoFaixa36Ate40);
    parametrosGráficoDTO.setTotalMunitoesMascullinoFaixa36Ate40(totalMunitoesMascullinoFaixa36Ate40);
    parametrosGráficoDTO.setTotalArmasFemininoFaixa36Ate40(totalArmasFemininoFaixa36Ate40);
    parametrosGráficoDTO.setTotalArmasMascullinoFaixa36Ate40(totalArmasMascullinoFaixa36Ate40);
}
```

- 
- Object Calisthenics
  - Prática

# Princípios do Clean Code (Revisão):

1. Nomes Significativos
2. Funções e Métodos
3. Comentários e Documentação
4. Formatação e Indentação
5. Evitar Duplicação de Código

# 1 - Nomes Significativos

- A importância de escolher nomes descritivos para variáveis, métodos e classes
- Evitar abreviações e nomes genéricos
- Exemplos de nomes significativos em Java

```
public class Customer {  
    private int customerId;  
    private String customerName;  
  
    public void placeOrder() {  
        // código aqui...  
    }  
}
```

```
public class A {  
    private int x;  
    private String z;  
  
    public void m() {  
        // código aqui...  
    }  
}
```

***Exemplo 1: Nomes sem sentido e com Significado***

## 2. Funções e Métodos:

- Aplicação do princípio da "Responsabilidade Única" (SRP) em Java
- Escrever funções curtas e com apenas uma responsabilidade
- Exemplos de refatoração de funções em Jav



## 2. Funções e Métodos:

```
1 public class ProductService {  
2     public Product calculateProductPriceAndSave(Product product, int quantity) {  
3         // Cálculos de preço do produto  
4         // Persistência do produto no banco de dados  
5         // Enviar e-mails de notificação  
6         // ...  
7         return product;  
8     }  
9 }
```

## Exemplo 2: Função com responsabilidade única

```
public class ProductService {  
    public Product calculateProductPrice(Product product, int quantity) {  
        // Cálculos de preço do produto  
        // ...  
        return product;  
    }  
  
    public void saveProduct(Product product) {  
        // Persistência do produto no banco de dados  
        // ...  
    }  
  
    public void sendNotificationEmail(Product product) {  
        // Enviar e-mails de notificação  
        // ...  
    }  
}
```

### 3. Comentários e Documentação:

- O papel dos comentários e documentação no código Java
- Priorizar um código autoexplicativo em vez de comentários excessivos
- Quando utilizar comentários para explicar lógicas complexas
- Exemplos de documentação Java com Javadoc

```
1 public double calculateRectangleArea(double length, double width) {  
2     // Multiplica o comprimento pelo largura para calcular a área do retângulo  
3     return length * width;  
4 }  
5
```

## Exemplo 3: Código autoexplicativo

```
/**  
 * Calcula a área de um retângulo.  
 * @param length Comprimento do retângulo.  
 * @param width Largura do retângulo.  
 * @return A área do retângulo.  
 */  
public double calculateRectangleArea(double length, double width) {  
    return length * width;  
}
```

## 4. Formatação e Indentação:

- A importância de seguir um padrão de formatação em Java
- Utilizar indentação adequada para melhorar a legibilidade
- Exemplos de formatação e indentação em Java

## 4. Formatação e Indentação:

- Exemplo 4: Formatação inconsistente java

```
public class OrderService {  
    public void processOrder(Order order) {  
        if (order != null){  
            // Processa o pedido  
            // ...  
        }  
    }  
}
```

```
public class OrderService {  
    public void processOrder(Order order) {  
        if (order != null) {  
            // Processa o pedido  
            // ...  
        }  
    }  
}
```

## 5. Evitar Duplicação de Código:

- Identificar e eliminar código duplicado em Java
- Utilizar métodos e classes para reutilização de código
- Exemplos de reutilização de código em Java

```
public void printMessage(boolean isError) {  
    if (isError) {  
        System.out.println("Error occurred!");  
    } else {  
        System.out.println("Action successful!");  
    }  
}  
  
public void logMessage(boolean isError) {  
    if (isError) {  
        System.err.println("Error occurred!");  
    } else {  
        System.out.println("Action successful!");  
    }  
}
```

## 6. Testabilidade e Testes Unitários:

```
public int calculateFactorial(int n) {  
    // Complex factorial calculation logic...  
    // ...  
}
```

```
public int calculateFactorial(int n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return n * calculateFactorial(n - 1);  
    }  
}
```

- Escrever código Java que seja facilmente testável
- A importância dos testes unitários para garantir a qualidade do código
- Exemplos de testes unitários em Java usando JUnit



## 7. Lidando com Exceções:

- Como lidar com exceções e erros de forma adequada em Java
- Evitar capturar exceções sem tratá-las corretamente
- Utilizar logs para rastrear problemas e facilitar a depuração em Java

```
try {  
    // Some operation that may throw an exception  
} catch (Exception ex) {  
    // Empty catch block  
}
```

```
try {  
    // Some operation that may throw an exception  
} catch (Exception ex) {  
    // Handle the exception (log, rethrow, etc.)  
}
```

## 8. Mantendo a Simplicidade:

- O princípio KISS (Keep It Simple, Stupid) aplicado em Java
- Evitar complexidade desnecessária e soluções complicadas
- Priorizar soluções simples e fáceis de entender em Java

```
public class CalculadoraComplexa {  
    public int adicionar(int a, int b) {  
        // Complexa lógica de adição...  
        // Muitas linhas de código aqui...  
        // ...  
    }  
  
    public int subtrair(int a, int b) {  
        // Complexa lógica de subtração...  
        // Mais linhas de código aqui...  
        // ...  
    }  
  
    public int multiplicar(int a, int b) {  
        // Complexa lógica de multiplicação...  
        // Muitas linhas de código aqui...  
        // ...  
    }  
  
    public int dividir(int a, int b) {  
        // Complexa lógica de divisão...  
        // Mais linhas de código aqui...  
        // ...  
    }  
}
```

```
1  public class CalculadoraSimples {
2      public int adicionar(int a, int b) {
3          return a + b;
4      }
5
6      public int subtrair(int a, int b) {
7          return a - b;
8      }
9
10     public int multiplicar(int a, int b) {
11         return a * b;
12     }
13
14     public int dividir(int a, int b) {
15         if (b == 0) {
16             throw new IllegalArgumentException("Não é possível dividir por zero.");
17         }
18         return a / b;
19     }
20 }
21
```

## 9. Refatoração Contínua:

- O que é refatoração e por que é importante em Java
- Refatorar o código Java para melhorar sua qualidade e legibilidade
- Como a refatoração ajuda na evolução do projeto sem introduzir bugs

```
// Long and messy function with duplicated code
public void processOrder(Order order) {
    // Lots of business logic and calculations here...
    // ...
    // More complex logic...
    // ...
    // Duplicated code block 1...
    // ...
    // More business logic...
    // ...
    // Duplicated code block 2...
    // ...
    // Even more business logic...
    // ...
}
```

```
public void processOrder(Order order) {
    // Refactored, more organized and clean code
    // Business logic split into smaller focused functions
    // Duplicated code extracted into reusable methods
    // ...
}
```

# Introdução ao Object Calisthenics com Spring Boot

- Objetivos:
- Compreender os princípios do Object Calisthenics.
- Aplicar as regras do Object Calisthenics para melhorar a qualidade e a legibilidade do código em projetos Spring Boot.
- Explorar exemplos práticos de como implementar as regras em diferentes cenários.


# Introdução:

- Object Calisthenics é um conjunto de diretrizes de programação que visam melhorar a qualidade do código, tornando-o mais claro, legível e manutenível. Ao aplicar essas regras, o desenvolvedor é incentivado a seguir práticas que levam a um código mais orientado a objetos, com menos acoplamento e melhor estruturado.

# Regra 1: Apenas um nível de identificação por método

- Métodos com múltiplos níveis de identificação tendem a ser complexos e difíceis de entender. Buscar dividir as responsabilidades e extrair métodos menores para manter um único nível de identificação aumenta a legibilidade do código.

```
// Ruim - Vários níveis de identificação
public void processarDados(List<Cliente> clientes) {
    for (Cliente cliente : clientes) {
        if (cliente.getTipo() == TipoCliente.PESSOA_FISICA) {
            if (cliente.getIdade() >= 18) {
                // Lógica de processamento
            }
        }
    }
}
```




```
1 // Bom - Apenas um nível de identificação
2 public void processarDados(List<Cliente> clientes) {
3     for (Cliente cliente : clientes) {
4         if (isClienteAdulto(cliente)) {
5             // Lógica de processamento
6         }
7     }
8 }
9
10 private boolean isClienteAdulto(Cliente cliente) {
11     return cliente.getTipo() == TipoCliente.PESSOA_FISICA && cliente.getIdade() >= 18;
12 }
```



## Regra 2: Não use a palavra-chave ELSE

- Evitar o uso de declarações "else" contribui para criar métodos mais independentes, facilitando a manutenção e compreensão do código.

```
// Ruim - Uso de "else"
public String getStatusPedido(Pedido pedido) {
    if (pedido.isPago()) {
        return "Pago";
    } else {
        return "Pendente";
    }
}
```




```
// Bom - Sem "else"
public String getStatusPedido(Pedido pedido) {
    if (pedido.isPago()) {
        return "Pago";
    }
    return "Pendente";
}
```

## Regra 3: Envolver tipos primitivos e strings em classes pequenas

- Ao invés de usar primitivos ou strings diretamente, encapsule-os em classes específicas para facilitar a manutenção e aplicar comportamentos relacionados.

```
// Ruim - Usando primitivos diretamente
public class Retângulo {
    private double altura;
    private double largura;

    public double calcularÁrea() {
        return altura * largura;
    }
}
```



```
// Bom - Encapsulando os primitivos em uma classe específica
public class Retângulo {
    private Altura altura;
    private Largura largura;

    public double calcularÁrea() {
        return altura.getValor() * largura.getValor();
    }
}
```

# Regra 4: Primeiras classes devem ser pequenas

```
// Ruim - Classe grande com muitas responsabilidades
public class Pedido {
    // Atributos e métodos relacionados a pedidos, pagamento, envio, etc...
}

// Bom - Classes pequenas e com responsabilidades específicas
public class Pedido {
    // Atributos e métodos relacionados a pedidos
}

public class Pagamento {
    // Atributos e métodos relacionados a pagamentos
}

public class Envio {
    // Atributos e métodos relacionados a envios
}
```

## Conclusão:

- O **Object Calisthenics** é uma abordagem que nos ajuda a escrever um código mais limpo, organizado e orientado a objetos. Ao aplicar essas regras em projetos Spring Boot, podemos melhorar a qualidade do código, tornando-o mais fácil de entender, manter e evoluir.
- Lembre-se de que a adoção dessas diretrizes não é uma regra rígida, mas sim uma sugestão para aprimorar a qualidade do código. Sempre é essencial avaliar cada situação e adaptar as práticas para atender às necessidades do projeto.

# Prática

- Vamos trabalhar colocar em prática o que aprendemos.

Pegue os projetos já desenvolvidos na disciplina e façam os ajustes para refletir o aprendizado deste dois dias.

- Caso não tenha realizado projeto da disciplina ou caso não encontre correções baixe a lista de exercícios
- <https://github.com/rodrigofujioka/boaspraticas>

# Prática

- Faça um fork do repositório X.
- Cria uma Branch no seu repositório