

# **CENTRO UNIVERSITÁRIO FACENS**

## **CURSOS TECNOLÓGICOS – AS406TSN1N2N3N4**



## **UX/UI E TESTES DE SOFTWARE**

# TÓPICOS DA AULA

## REFATORAÇÃO

- INTRODUÇÃO
- EXEMPLO

## O QUE TESTAR?

- REQUISITOS FUNCIONAIS
- TESTES FUNCIONAIS

## QUANDO TESTAR?

- TESTE AUTOMATIZADO UNITÁRIO
- EXERCÍCIO
- REPOSITÓRIO REMOTO

# REFATORAÇÃO

A refatoração é uma prática comum na engenharia de software e é geralmente realizada durante o ciclo de desenvolvimento do software. Ela pode ser aplicada para corrigir problemas de design, eliminar duplicação de código, melhorar a eficiência, facilitar a compreensão do código e torná-lo mais fácil de testar.

A refatoração é uma prática importante para manter a qualidade do código ao longo do tempo e é frequentemente associada a metodologias ágeis de desenvolvimento de software, como o Extreme Programming (XP) e o Scrum.

É importante ressaltar que, embora a refatoração altere a estrutura interna do código, ela não deve modificar o comportamento externo do software, sendo essencial realizar testes automatizados para garantir que as alterações não introduziram regressões ou novos bugs.



# REFATORAÇÃO

O termo "refatoração" foi originalmente usado por William Opdyke em sua tese de doutorado, defendida em 1992 na Universidade de Illinois, nos Estados Unidos. Opdyke descreveu técnicas para reestruturar o código de forma incremental para melhorar sua legibilidade e manutenibilidade.

Apesar de sua origem acadêmica, a prática de refatoração foi adotada de forma mais ampla na indústria de desenvolvimento de software após a publicação do livro de Martin Fowler (1999). O livro de Fowler forneceu uma abordagem sistemática e uma coleção de padrões de refatoração com exemplos concretos, o que ajudou a popularizar a ideia e a torná-la acessível aos desenvolvedores de software.



# REFATORAÇÃO

Alguns exemplos de refatorações comuns incluem:

- **Extrair método:** Consiste em identificar blocos de código que realizam uma função específica e extrair esse bloco para um método separado, tornando o código mais modular e legível.
- **Renomear variáveis ou métodos:** Renomear elementos do código para refletir melhor sua funcionalidade ou propósito, tornando o código mais descritivo e fácil de entender.
- **Consolidar duplicações:** Identificar trechos de código duplicados e reestruturá-los em uma única função ou classe, reduzindo a redundância e facilitando futuras alterações.
- **Dividir classes ou métodos grandes:** Dividir classes ou métodos muito grandes em unidades menores e mais gerenciáveis, melhorando a organização do código e tornando-o mais fácil de entender e manter.
- **Simplificar condicionais ou loops:** Simplificar estruturas condicionais ou loops complexos, tornando o código mais legível e menos propenso a erros.

# REFATORAÇÃO

```
public class CalculadoraCompra {  
  
    public double calcularPrecoTotal(double precoUnitario, int quantidade) {  
        double precoTotal = precoUnitario * quantidade;  
  
        // Aplica um desconto de 10% se o preço total for superior a 100  
        if (precoTotal > 100) {  
            precoTotal = precoTotal * 0.9;  
        }  
  
        return precoTotal;  
    }  
}
```



# REFATORAÇÃO

```
public class CalculadoraCompra {  
  
    public double calcularPrecoTotal(double precoUnitario, int quantidade) {  
        double precoTotal = precoUnitario * quantidade;  
  
        aplicarDesconto(precoTotal);  
  
        return precoTotal;  
    }  
  
    private void aplicarDesconto(double precoTotal) {  
        // Aplica um desconto de 10% se o preço total for superior a 100  
        if (precoTotal > 100) {  
            precoTotal = precoTotal * 0.9;  
        }  
    }  
}
```



# DOCUMENTAÇÃO

A importância da documentação no código-fonte em contexto atual, em produção de software 75% do custo de software está na manutenção. Sendo que desenvolvedores gastam 50-70% do tempo entendendo código existente. Além disso, a rotatividade em equipes faz com que um projeto tenha uma documentação para que exista um "plano de continuidade" do projeto.

```
/**
 * Calcula o score de crédito do cliente baseado em múltiplos fatores
 * usando machine learning e dados históricos.
 *
 * @param cpf String formatada do CPF (XXX.XXX.XXX-XX)
 * @param rendaMensal valor em reais (deve ser > 0)
 * @param historico lista de transações dos últimos 24 meses
 * @return Score entre 0-1000 onde ≥700 indica baixo risco
 * @throws IllegalArgumentException se CPF for inválido ou renda ≤ 0
 * @since 2.1
 * @author Time de Risk Analytics
 * @see #validarCPF(String) para verificação de formato
 */
public int calcularScoreCredito(String cpf, double rendaMensal,
                                List<Transacao> historico) {
    // Implementação complexa...
```

```
/**
 * WEBHOOK - Endpoint para notificações de pagamento PIX
 *
 * Chamado automaticamente pelo Banco Central quando:
 * - Pagamento é confirmado
 * - Pagamento expira
 * - Ocorre erro na transação
 *
 * ⚠️ IMPORTANTE: Este endpoint DEVE responder em < 2 segundos
 *
 * @param notificacao DTO com status e metadados da transação
 * @return HttpStatus.ACCEPTED (202) mesmo em caso de erro interno
 * @throws WebhookException apenas para timeout do BC
 */
@PostMapping("/webhook/pix")
public ResponseEntity<Void> webhookPix(@RequestBody NotificacaoPix notificacao) {
    // Time de Frontend sabe exatamente como integrar
```

# DOCUMENTAÇÃO

Linguagem	Ferramenta Principal	Formato de Saída	Geração	Ferramentas Complementares
Java	JavaDoc	HTML, PDF	Automática	Maven Site Plugin, AsciiDoc, Spring REST Docs
Python	Docstrings (Sphinx)	HTML, PDF, ePub	Semi-automática	MkDocs, pydoc, FastAPI autodoc
JavaScript/TypeScript	JSDoc/TypeDoc	HTML, JSON	Automática	ESLint, VuePress, Storybook
C#	XML Documentation	HTML, CHM	Automática	Sandcastle, DocFX, Swashbuckle
Go	godoc	HTML, Web Server	Automática	pkgsite, Go Report Card
Rust	rustdoc	HTML	Automática	cargo doc, mdBook
PHP	PHPDoc	HTML, PDF	Automática	PHPStan, Psalm, ApiGen
Ruby	RDoc, YARD	HTML, RI	Automática	Sphinx Ruby, GitHub Wikis
Swift	Swift DocC	HTML, Xcode Docs	Automática	Jazzy, Apple's DocC
Kotlin	Dokka	HTML, Markdown	Automática	KDoc, Kotlin Multiplatform
Scala	Scaladoc	HTML	Automática	ScalaDoc, tut
Dart	dartdoc	HTML	Automática	Flutter Doc, <a href="https://pub.dev">pub.dev</a>

# EXERCÍCIO

## Contexto da Atividade

Você foi contratado como desenvolvedor júnior pela startup "**FinançApp**", que está desenvolvendo um aplicativo mobile para controle de finanças pessoais. A equipe precisa de um módulo básico de calculadora para funcionalidades como:

- Cálculo de orçamento mensal
- Divisão de despesas em grupos
- Projeção de economias
- Cálculo de juros simples

## Cenários de Uso no Aplicativo:

*"Os usuários do FinançApp precisam calcular rapidamente quanto podem gastar por dia baseado no salário mensal, dividir contas de restaurante entre amigos, ou projetar quanto economizarão em 12 meses."*

# EXERCÍCIO

## **Tarefa: Implementar o Núcleo da Calculadora**

Você deve desenvolver a classe Calculadora que servirá como motor de cálculos para todas as funcionalidades financeiras do aplicativo.

### **Requisitos Funcionais:**

**Operações Básicas:** A calculadora deve realizar as 4 operações fundamentais:

Adição (+)

Subtração (-)

Multiplicação (\*)

Divisão (/)

### **Tratamento de Erros:**

Bloquear divisão por zero com mensagem informativa

Alertar sobre operações inválidas

Garantir que nenhum cálculo quebre o aplicativo

# EXERCÍCIO

## Especificações Técnicas:

**Classe:** Calculadora

**Pacote:** calculadora

**Método principal:** calc(int a, int b, String op)

**Atributo:** r (resultado)

**Retorno:** sempre um int com o resultado

## Critérios de Aceitação:

- **Funcionalidade:** Todas as operações básicas funcionam corretamente
- **Robustez:** Não permite divisão por zero e trata operações inválidas
- **Usabilidade:** Fornece feedback claro para o usuário
- **Manutenibilidade:** Código limpo e bem estruturado

# EXERCÍCIO

Crie um projeto na linguagem Java com o título: projeto\_calculadora; Dentro do projeto crie um pacote com título: calculadora;

No pacote crie uma classe calculadora.java e dentro digite o seguinte, conforme a figura direita;

O seguinte código deverá ser testado de forma unitária, refatorado e documentado.

Além disso, devemos criar um repositório remoto e postar o mesmo com todos os procedimentos acima realizados.

```
scr > calculadora > J Calculadora.java > Calculadora > calc(int, int, String)
1  package calculadora;
2
3  public class Calculadora {
4
5      public int r = 0;
6
7      public int calc(int a, int b, String op) {
8
9          //Calculadora
10         if (op.equals(anObject: "+")) {
11             r = a + b;
12         } else if (op.equals(anObject: "-")) {
13             r = a - b;
14         } else if (op.equals(anObject: "*")) {
15             r = a * b;
16         } else if (op.equals(anObject: "/")) {
17             if (b != 0) {
18                 r = a / b;
19             } else {
20                 System.out.println(x: "Erro: divisao por zero, operação irregular");
21                 r = 0;
22             }
23         } else {
24             System.out.println(x: "Operacao invalida");
25             r = 0;
26         }
27
28         System.out.println("Resultado = " + r);
29         return r;
30     }
31 }
32
```

# EXERCÍCIO

Para testar o projeto de forma que após uma modificação seja feita e que garanta a confiabilidade do projeto criaremos uma classe de teste.

Para isso crie uma classe na raiz do pacote com o título `TesteCalculadora`, esta classe ao ser compilada testará o funcionamento da classe `Calculadora`. Isso seria um processo de teste unitário e aplicado de forma segura.

Veja que nos comentários temos informações dos dados esperados de saída, ao terminar de digitar o código na figura ao lado, compile seu código e verifique o que é apresentado no console.

```
1 package calculadora;
2
3 public class TesteCalculadora {
4
5     Run | Debug
6     public static void main(String[] args) {
7         Calculadora calc = new Calculadora();
8
9         // Testes rápidos
10        calc.calc(a: 2, b: 3, op: "+"); // esperado: 5
11        calc.calc(a: 10, b: 4, op: "-"); // esperado: 6
12        calc.calc(a: 3, b: 5, op: "*"); // esperado: 15
13        calc.calc(a: 8, b: 2, op: "/"); // esperado: 4
14        calc.calc(a: 8, b: 0, op: "/"); // divisao por zero
15        calc.calc(a: 5, b: 5, op: "x"); // operação inválida
16    }
17 }
```

# EXERCÍCIO

Com o código-fonte criado, testado, devemos criar um repositório remoto do projeto. Ao termino do crie uma Branch nomeada Refatoracao;

Nesta Branch modificaremos o projeto original refatorando, ao termino devemos testar novamente para avaliar o funcionamento do código e verificar seu funcionamento. Para isso seguiremos os próximos passos solicitados nos slides a seguir.

# EXERCÍCIO

O primeiro passo é o isolamento das dos métodos criados para execução dos cálculos para classes isolando seu funcionamento e aprimorando o código para manutenções futuras.

Além disso, daremos o devido tratamento para as exceções, conforme a figura ao lado.

```
1 package calculadora;
2
3 public class Calculadora {
4
5     // métodos puros e simples
6     public int somar(int a, int b) {
7         return a + b;
8     }
9
10    public int subtrair(int a, int b) {
11        return a - b;
12    }
13
14    public int multiplicar(int a, int b) {
15        return a * b;
16    }
17
18    public int dividir(int a, int b) {
19        if (b == 0) {
20            throw new IllegalArgumentException(s: "Divisão por zero não é permitida");
21        }
22        return a / b;
23    }
24
25    public int calcular(int a, int b, String operador) {
26        return switch (operador) {
27            case "+" -> somar(a, b);
28            case "-" -> subtrair(a, b);
29            case "*" -> multiplicar(a, b);
30            case "/" -> dividir(a, b);
31            default -> throw new IllegalArgumentException("Operador inválido: " + operador);
32        };
33    }
34 }
35
```

# EXERCÍCIO

Ajustaremos também nosso código de teste. Para isso na classe TesteCalculadora, altera o código para o modelo apresentado na figura ao lado.

Após a alteração compile novamente o código e veja se os resultados apresentados.

Tire um print do resultado demonstrado no terminal e poste no README.MD da branch

```
1 package calculadora;
2
3 public class TesteCalculadora {
4
5     Run | Debug
6     public static void main(String[] args) {
7         Calculadora calc = new Calculadora();
8
9         System.out.println(calc.calcular(a: 2, b: 3, operador: "+"));
10        System.out.println(calc.calcular(a: 10, b: 4, operador: "-"));
11        System.out.println(calc.calcular(a: 3, b: 5, operador: "**"));
12        System.out.println(calc.calcular(a: 8, b: 2, operador: "/"));
13
14        try {
15            System.out.println(calc.calcular(a: 8, b: 0, operador: "/")); // exceção
16        } catch (IllegalArgumentException e) {
17            System.out.println(e.getMessage());
18        }
19
20        try {
21            System.out.println(calc.calcular(a: 5, b: 5, operador: "x")); // exceção
22        } catch (IllegalArgumentException e) {
23            System.out.println(e.getMessage());
24        }
25    }
26 }
```

# EXERCÍCIO

Uma vez testado realizaremos o processo de documentação, conforme vimos anteriormente este processo não é somente utilizar os comentários utilizando a barra dupla (//). Devemos utilizar blocos de informações que esclareçam para o leitor sobre o que o código está fazendo. Para isso devemos primeiro fazer os devidos comentários no projeto, conforme o exemplo que estão nos próximos slides. Uma vez que realizar os devidos comentários devemos gerar os comentários para a classe de teste.

No fim, devemos gerar o javadoc do projeto, para isso acesse a pasta src que está dentro do seu projeto e execute pelo terminal o comando: `javadoc -d ../docs *.java` (caso utilize o VSCode), se for utilizado a IDE Eclipse siga as opções que estão no final deste slide.

Isso irá gerar uma pasta com os documentos em html, caso acesse o index.html no navegador e verá que teremos a documentação em html, conforme a seguir.

# EXERCÍCIO

```
3  /**
4   * <p><strong>Projeto Calculadora</strong></p>
5   *
6   * <p>Esta classe fornece operações matemáticas básicas como soma, subtração,
7   * multiplicação e divisão, além de um método principal que seleciona a operação
8   * desejada com base em um operador informado pelo usuário.</p>
9   * <p>Esse projeto é utilizado para demonstrar:</p>
10  * <ul>
11  *   <li>Técnicas de revisão estática (caixa branca)</li>
12  *   <li>Técnicas de revisão (caixa branca)</li>
13  *   <li>Aplicação de refatoração em código Java</li>
14  *   <li>Documentação com Javadoc</li>
15  * </ul>
16  *
17  * <p>Após a refatoração, os métodos desta classe são <strong>métodos puros</strong>,
18  * sem efeitos colaterais, permitindo maior legibilidade e facilitando testes unitários.</p>
19  *
20  * @author Seu Nome
21  * @version 1.0
22  */
23 public class Calculadora {
24
25     /**
26      * Soma dois números inteiros.
27      *
28      * @param a primeiro operando
29      * @param b segundo operando
30      * @return o resultado da soma de {@code a} e {@code b}
31      */
32     public int somar(int a, int b) {
33         return a + b;
34     }
35
36     /**
37      *
38      *
39      * Preencher com o devido comentário
40      */
41     public int subtrair(int a, int b) {
42         return a - b;
43     }
44
45     /**
46      *
47      *
48      * Preencher com o devido comentário
49      */
50     public int multiplicar(int a, int b) {
51         return a * b;
52     }
53 }
```

```
53
54 /**
55 *
56 *
57 * Preencher com o devido comentário
58 */
59 public int dividir(int a, int b) {
60     if (b == 0) {
61         throw new IllegalArgumentException(s: "Divisão por zero não é permitida.");
62     }
63     return a / b;
64 }
65
66 /**
67  * <p>Executa a operação matemática solicitada com base no operador informado.</p>
68  *
69  * <p>Operadores aceitos:</p>
70  * <ul>
71  *   <li>"+":Soma</li>
72  *   <li>"-":Subtração</li>
73  *   <li>"*":Multiplicação</li>
74  *   <li>"/":Divisão</li>
75  * </ul>
76  *
77  * <p>Qualquer operador inválido resulta em exceção.</p>
78  *
79  * @param a primeiro operando
80  * @param b segundo operando
81  * @param operador símbolo da operação desejada
82  * @return o resultado da operação correspondente ao operador informado
83  *
84  * @throws IllegalArgumentException se o operador não for um símbolo válido
85  */
86 public int calcular(int a, int b, String operador) {
87     return switch (operador) {
88         case "+" -> somar(a, b);
89         case "-" -> subtrair(a, b);
90         case "*" -> multiplicar(a, b);
91         case "/" -> dividir(a, b);
92         default -> throw new IllegalArgumentException("Operador inválido: " + operador);
93     };
94 }
95 }
96
```

# EXERCÍCIO

PACKAGE

CLASS

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

## Package calculadora

package calculadora

Classes

Class	Description
Calculadora	Projeto Calculadora

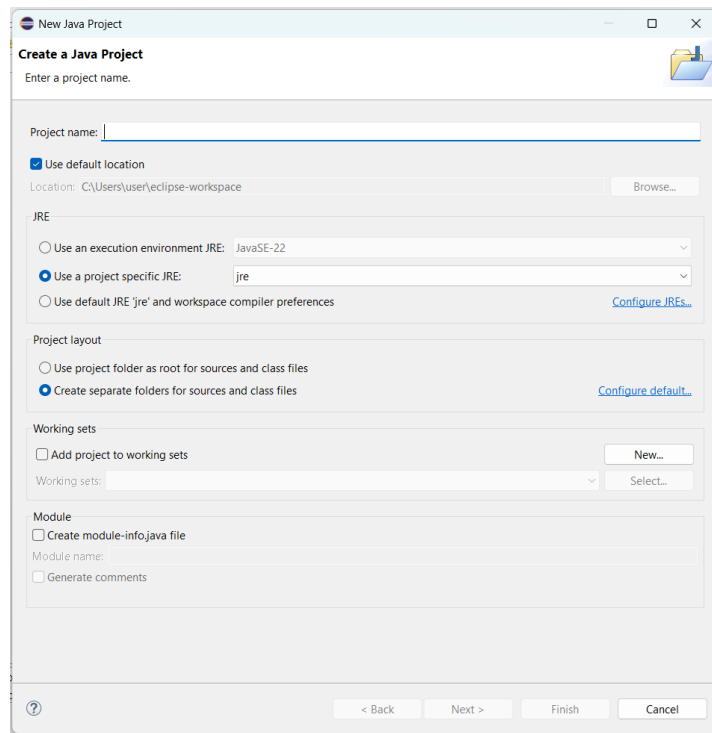
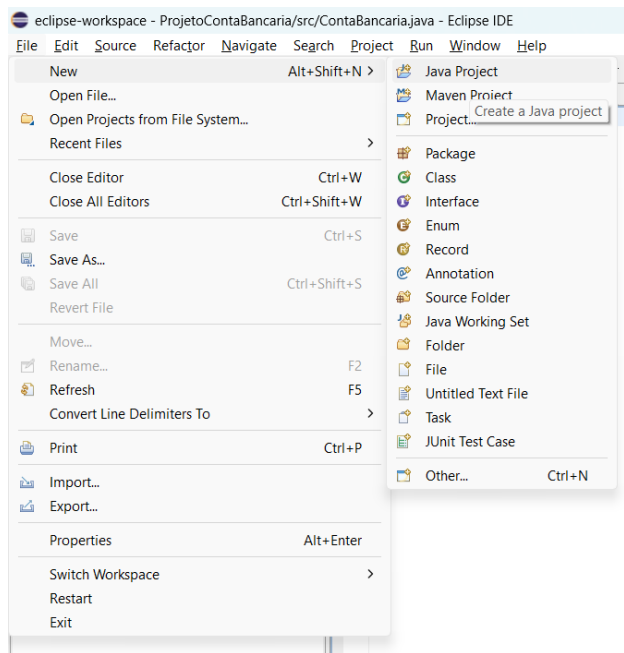
# EXEMPLO DO USO DO JAVADOC

Neste exemplo, criaremos um projeto no **Eclipse IDE** para gerenciar uma conta bancária, utilizando o **JavaDoc** para documentar todas as partes do código. Seguindo este guia, você aprenderá a comentar adequadamente seu código e gerar a documentação usando JavaDoc.

## Passo 1: Criar um novo projeto no Eclipse

- Abra o Eclipse IDE. Vá para File > New > Java Project.
- Dê o nome ao projeto, por exemplo: ProjetoContaBancaria.
- Clique em Finish.

# EXEMPLO DO USO DO JAVADOC



# EXEMPLO DO USO DO JAVADOC

## **Passo 2: Criar a estrutura do projeto**

- Clique com o botão direito no nome do projeto no Package Explorer.
- Vá para New > Package e crie um pacote chamado br.com.projeto.contabancaria.
- Clique com o botão direito no pacote, vá para New > Class e crie as seguintes classes:
  - ContaBancaria.java
  - Main.java

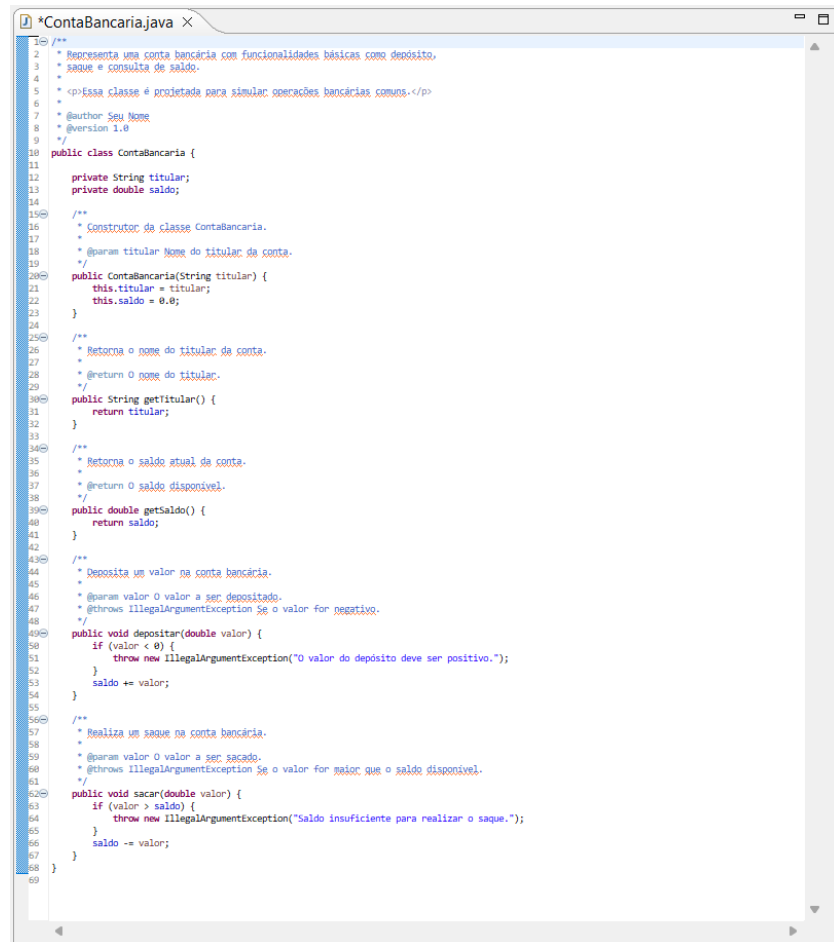
# EXEMPLO DO USO DO JAVADOC

Esta classe gerencia as operações básicas de uma conta bancária, como depósito, saque e consulta de saldo.

Cada trecho do código contém grandes blocos de comentários, mas o ponto que devemos ter atenção fica por conta de ser comandos dados no comentário.

Veja que na linha 7 e 8 utilizamos as anotações `@author` e `@version`.

Essas anotações serão utilizadas para construir a documentação no JAVADOC.



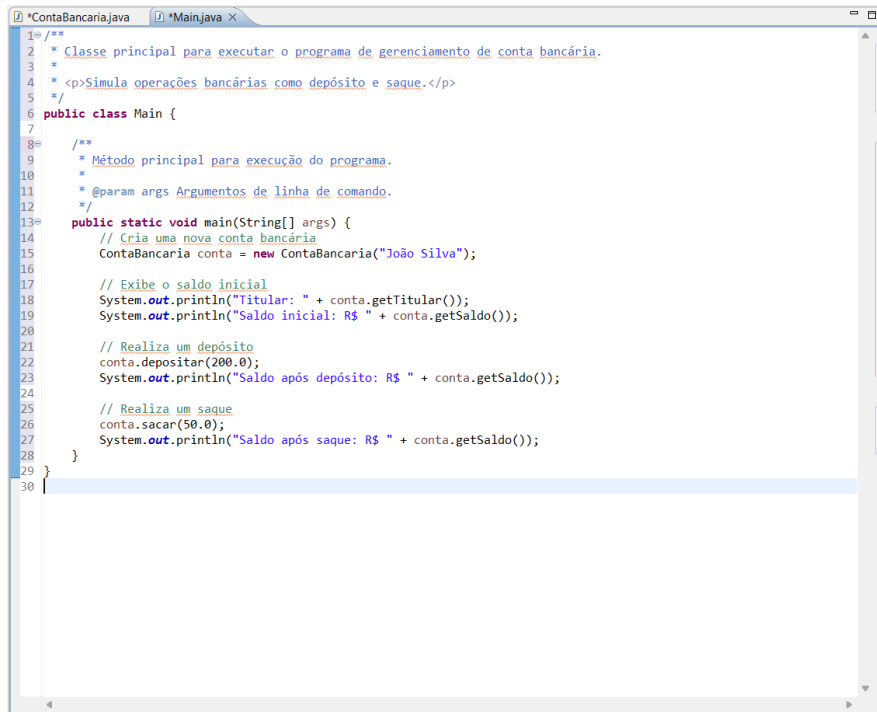
```
1 //
2 * Representa uma conta bancária com funcionalidades básicas como depósito,
3 * saque e consulta de saldo.
4
5 * <p>Essa classe é projetada para simular operações bancárias comuns.</p>
6 *
7 * @author Seu Nome
8 * @version 1.0
9 */
10 public class ContaBancaria {
11
12     private String titular;
13     private double saldo;
14
15     /**
16      * Construtor da classe ContaBancaria.
17      *
18      * @param titular Nome do titular da conta.
19      */
20     public ContaBancaria(String titular) {
21         this.titular = titular;
22         this.saldo = 0.0;
23     }
24
25     /**
26      * Retorna o nome do titular da conta.
27      *
28      * @return O nome do titular.
29      */
30     public String getTitular() {
31         return titular;
32     }
33
34     /**
35      * Retorna o saldo atual da conta.
36      *
37      * @return O saldo disponível.
38      */
39     public double getSaldo() {
40         return saldo;
41     }
42
43     /**
44      * Deposita um valor na conta bancária.
45      *
46      * @param valor O valor a ser depositado.
47      * @throws IllegalArgumentException Se o valor for negativo.
48      */
49     public void depositar(double valor) {
50         if (valor < 0) {
51             throw new IllegalArgumentException("O valor do depósito deve ser positivo.");
52         }
53         saldo += valor;
54     }
55
56     /**
57      * Realiza um saque na conta bancária.
58      *
59      * @param valor O valor a ser sacado.
60      * @throws IllegalArgumentException Se o valor for maior que o saldo disponível.
61      */
62     public void sacar(double valor) {
63         if (valor > saldo) {
64             throw new IllegalArgumentException("Saldo insuficiente para realizar o saque.");
65         }
66         saldo -= valor;
67     }
68 }
69
```

# EXEMPLO DO USO DO JAVADOC

Completando nosso projeto temos a classe Main.java.

Em especial a Eclipse IDE ainda diferencia comentários simples dos grandes blocos de comentários.

Veja que na linha 14 temos um comentário simples em verde e na linha 1 a linha 5 temos um bloco de comentários em azul.

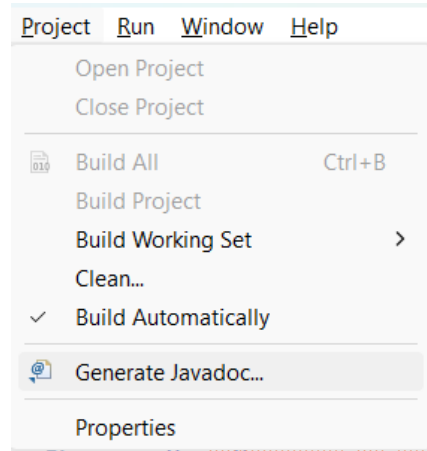
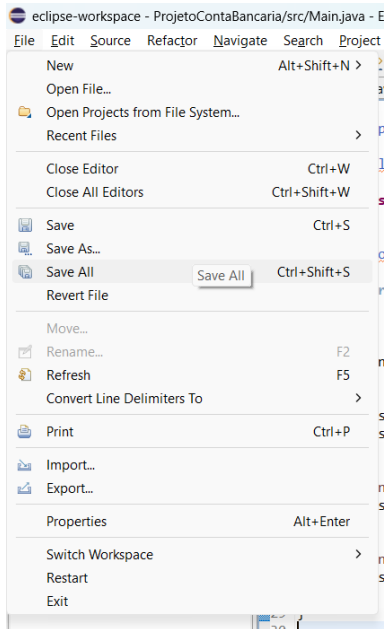


```
1 /**
2  * Classe principal para executar o programa de gerenciamento de conta bancária.
3  *
4  * <p>Simula operações bancárias como depósito e saque.</p>
5  */
6 public class Main {
7
8     /**
9     * Método principal para execução do programa.
10    *
11    * @param args Argumentos de linha de comando.
12    */
13    public static void main(String[] args) {
14        // Cria uma nova conta bancária
15        ContaBancaria conta = new ContaBancaria("João Silva");
16
17        // Exibe o saldo inicial
18        System.out.println("Titular: " + conta.getTitular());
19        System.out.println("Saldo inicial: R$ " + conta.getSaldo());
20
21        // Realiza um depósito
22        conta.depositar(200.0);
23        System.out.println("Saldo após depósito: R$ " + conta.getSaldo());
24
25        // Realiza um saque
26        conta.sacar(50.0);
27        System.out.println("Saldo após saque: R$ " + conta.getSaldo());
28    }
29 }
30
```

# EXEMPLO DO USO DO JAVADOC

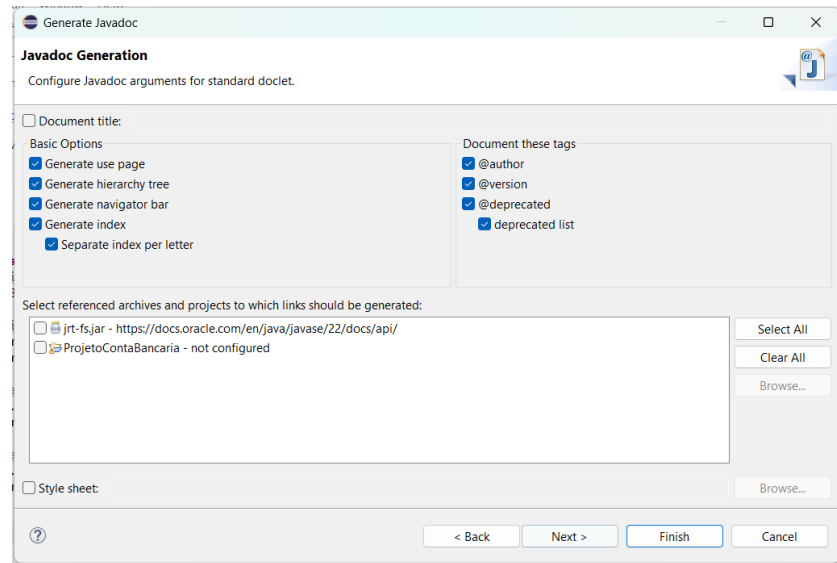
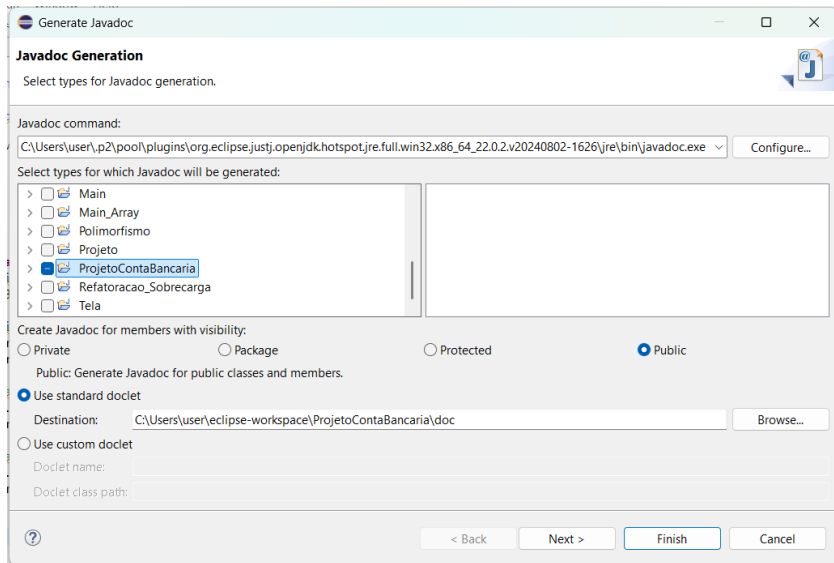
## Passo 3: Como gerar a documentação JavaDoc no Eclipse

- Certifique-se de que o código está salvo (imagem a esquerda).
- Vá para o menu Project > Generate Javadoc (imagem a direita).



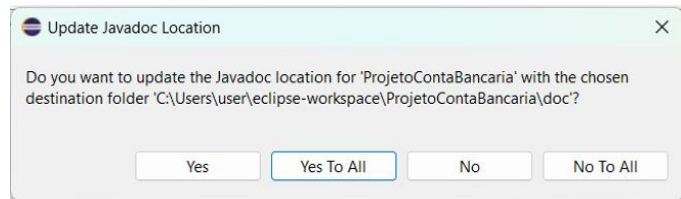
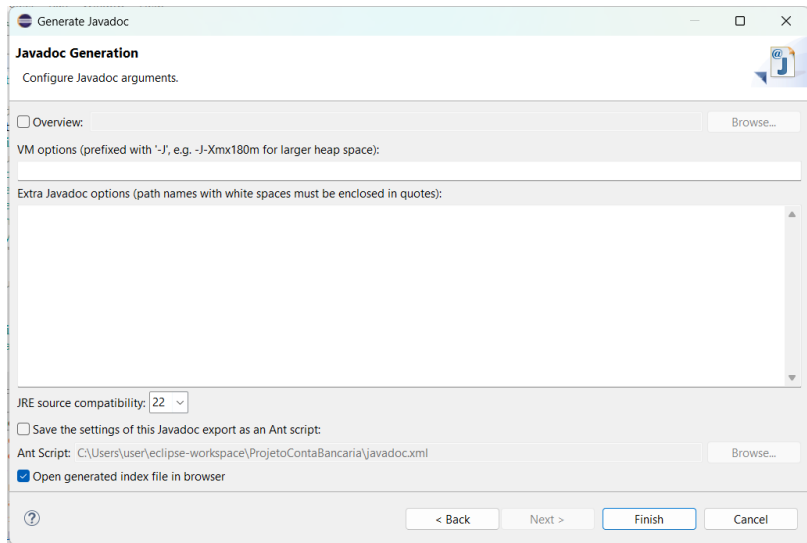
# EXEMPLO DO USO DO JAVADOC

- Verifique que a opção do seu projeto está selecionada e clique em Next (imagem esquerda).
- Nesta janela observe que as anotações serão utilizadas para identificar os devidos comentários no JAVADOC (imagem direita), após isso clique em NEXT.



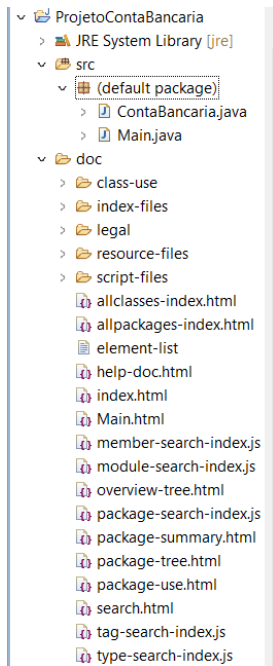
# EXEMPLO DO USO DO JAVADOC

- Selecione a opção “Open generates index file in browser”, nela teremos ao termino do processo a abertura da documentação no navegador (imagem a direita).
- Para completar clique em NEXT e por fim selecione a opção Yes All, isso confirmará a criação da pasta doc, nela será mantida toda a documentação do projeto.



# EXEMPLO DO USO DO JAVADOC

- Em seu navegador, veja que abrirá o documento criado, por ele você poderá verificar os comentários feitos no projeto. Além disso, na pasta do seu projeto haverá a pasta doc com os arquivos criados.
- No endereço de navegação estará presente o local do seu index de acesso.



# EXEMPLO DO USO DO JAVADOC

- Podemos conferir o conteúdo das classe e seus comentários na integra, conforme a imagem que apresenta da Classe ContaBancaria.

PACKAGE

CLASS

USE

TREE

INDEX

HELP

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

SEARCH

## Class ContaBancaria

java.lang.Object<sup>Ⓓ</sup>  
ContaBancaria

---

```
public class ContaBancaria
extends ObjectⒹ
```

Representa uma conta bancária com funcionalidades básicas como depósito, saque e consulta de saldo.

Essa classe é projetada para simular operações bancárias comuns.

Version:  
1.0

Author:  
Seu Nome

### Constructor Summary

Constructors

Constructor	Description
ContaBancaria(String <sup>Ⓓ</sup> titular)	Construtor da classe ContaBancaria.

### Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	depositar(double valor)	Deposita um valor na conta bancária.
double	getSaldo()	Retorna o saldo atual da conta.
String <sup>Ⓓ</sup>	getTitular()	Retorna o nome do titular da conta.
void	sacar(double valor)	Realiza um saque na conta bancária.

Methods inherited from class java.lang.Object<sup>Ⓓ</sup>

equals<sup>Ⓓ</sup>, getClass<sup>Ⓓ</sup>, hashCode<sup>Ⓓ</sup>, notify<sup>Ⓓ</sup>, notifyAll<sup>Ⓓ</sup>, toString<sup>Ⓓ</sup>, wait<sup>Ⓓ</sup>, wait<sup>Ⓓ</sup>, wait<sup>Ⓓ</sup>

# EXERCÍCIO

Serão aceitos somente os repositórios remotos com a data do dia da atividade e com a entrega presencial do aluno;

Está atividade é a segunda parte do teste de caixa branca e completa as atividades finais individuais;

Durante a execução da atividade não é permitido comunicação entre os alunos e o uso do celular;

Está atividade é individual e deve ser entregue até o término desta aula.1

# PERGUNTAS?



[daniel.ohata@facens.br](mailto:daniel.ohata@facens.br)

**MUITO OBRIGADO!!!!**



**daniel.ohata@facens.br**

# Referências Bibliográficas

- CARPINETT, Luiz Cesar Ribeiro; GEROLAMO, Mateus Cecílio. **Gestão da Qualidade ISO 9001**. Atlas, 2016.
- MACHADO, Felipe Nery Rodrigues. **Análise e Gestão de Requisitos de Software. Onde Nasce os Sistemas**. 3 ed. Érica, 2015.
- PRESSMAN, Roger S. MAXIM, Bruce R. **Engenharia de Software - Uma Abordagem Profissional**. 8.ed. Porto Alegre: Amgh Editora, 2016. 968p. ISBN 9788580555332.
- CORDELLI, Rosa Lantmann; LAUREANO, Marcos Aurelio Pchek. **Fundamentos de Software - Desempenho de Sistemas Computacionais**. Erica, 2014. ISBN Digital: 9788536519234.
- MACHADO, Felipe Nery Rodrigues. **Banco de Dados - Projeto e Implementação**. Erica, 2014. ISBN Digital: 9788536518961.
- MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. **Algoritmos - Lógica para Desenvolvimento de Programação de Computadores**. Erica, 2016. ISBN Digital: 9788536518657.
- MANZANO, José Augusto N. G.; LOURENÇO, André Evandro; MATOS, Edivaldo. **Algoritmos - Técnicas de Programação**. Erica, 2016. ISBN Digital: 9788536518664.
- STAIR, Ralph M.; REYNOLDS, George W. **Princípios de Sistemas de Informação**. 11.ed. Cengage Learning Editores, 2016. ISBN Digital: 9788522124107.