



Quem se prepara, não para.



# Linguagens de Programação

3º período

Prof. Dr. João Paulo Aramuni

# Sumário

- Java
  - Boas práticas de programação

# Sumário

- Java
  - Boas práticas de programação

- Boas práticas de programação
  - 1) Indentação
    - Indentação refere-se à prática de adicionar espaços ou tabs (tabulações) no início de uma linha de código para indicar sua posição hierárquica em relação a outras linhas.
    - Em outras palavras, é a maneira de **formatar visualmente o código para torná-lo mais legível e organizado**, facilitando a compreensão da estrutura do programa.
    - Na linguagem Python, por exemplo, a indentação é extremamente importante e pode fazer o código não funcionar corretamente se não for utilizada de forma adequada.

- Boas práticas de programação
  - 1) Indentação

```
1 package br.com.javamagazine;
2 class Vendas implements InterfaceVendas{
3     public String nomeCliente;
4     protected String descProduto;
5     private double valor=50;
6     public double altera_valor(double valor){
7         this.valor = valor;
8         return valor;
9     }
10    public void imprime(double valor){
11        if (valor>100)
12            JOptionPane.showMessageDialog(null,"Valor acima do permitido");
13        else JOptionPane.showMessageDialog(null,"Valor="+valor);
14    }
15 }
```

Código sem indentação - inadequado

```
1 package br.com.javamagazine;
2
3     import javax.swing.JOptionPane;
4
5     class Vendas implements InterfaceVendas {
6
7         public String nomeCliente;
8         protected String descProduto;
9         private double valor = 50;
10
11         public double alteraValor(double valor){
12             this.valor = valor;
13             return valor;
14         }
15
16         public void imprimeValor(double valor){
17             if (valor > 100)
18                 JOptionPane.showMessageDialog(null,"Valor acima do permitido");
19             else
20                 JOptionPane.showMessageDialog(null,"Valor="+valor);
21         }
22
23     }
```

Código com indentação - boa prática

- Boas práticas de programação
  - 1) Indentação
    - Atalhos de teclado no Eclipse:
      - Windows / Linux:
        - Indentar uma linha: Pressione Ctrl + Shift + F
        - Indentar um trecho selecionado: Pressione Ctrl + I
      - Mac:
        - Indentar uma linha: Pressione Cmd + Shift + F
        - Indentar um trecho selecionado: Pressione Cmd + I

- Boas práticas de programação
  - 2) Comentários e documentação
    - Explicar o algoritmo ou a lógica usada, mostrando o objetivo de uma variável, método, classe, etc.
    - Documentar o projeto, descrevendo a especificação do código e regras de negócio.
      - Em geral os projetos de desenvolvimento são muito mal documentados.
    - Por meio de uma ferramenta chamada **javadoc** é possível gerar documentação baseada em tags específicas que você coloca no seu código fonte.



# Java

- Boas práticas de programação
  - 2) Comentários e documentação
    - javadoc

Comentário javadoc  
Iniciando com /\*\*

Comentário comum  
// (linha) ou /\*\*/ (bloco)

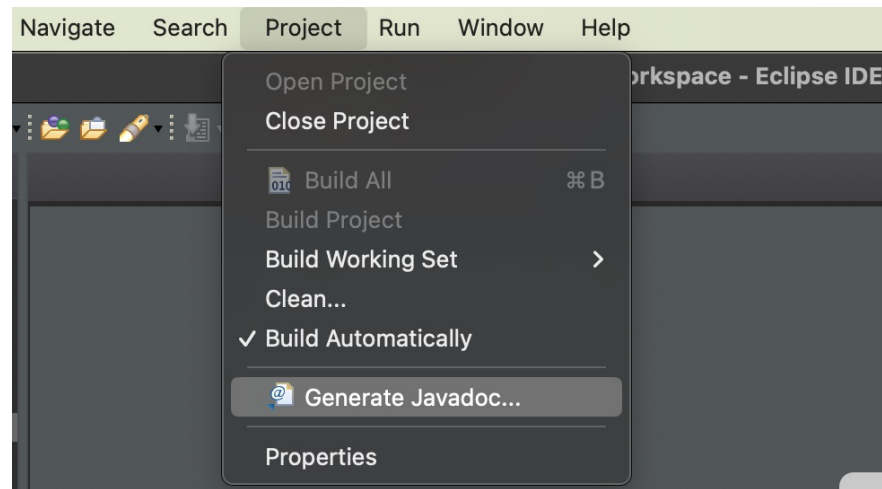
```
1  /*
2   * Procure sempre comentar no início do arquivo fonte colocando informações como
3   * Nome da Classe, Data da Criação
4   * Todos os direitos reservados para Nome da Empresa e o endereço desta completo
5   * Outras informações que achar necessário.
6   * Se você não for gerar documentação ou se um comentário não é adequado para isso,
7   * também pode colocá-lo aqui
8   */
9
10 package br.com.devmedia;
11
12 import javax.swing.JOptionPane;
13
14 /**
15  * A informação que você colocar aqui irá para a documentação gerada através do javadoc.
16  * @version 1.0
17  * @author Neri Aldoir Neitzke - Universidade Ulbra
18  */
19
20 public class CalcMedia {
21     public static void main(String args[]){
22         float nota1, nota2, media;
23         nota1 = 5;
24         nota2 = 7;
25         media = (nota1 + nota2) / 2; //calcula a media do aluno
26         if (media >= 6)
27             JOptionPane.showMessageDialog(null,"Aprovado com média " + media);
28         else
29             JOptionPane.showMessageDialog(null,"Reprovado com média " + media);
30     }
```

- Boas práticas de programação
  - 2) Comentários e documentação
    - javadoc



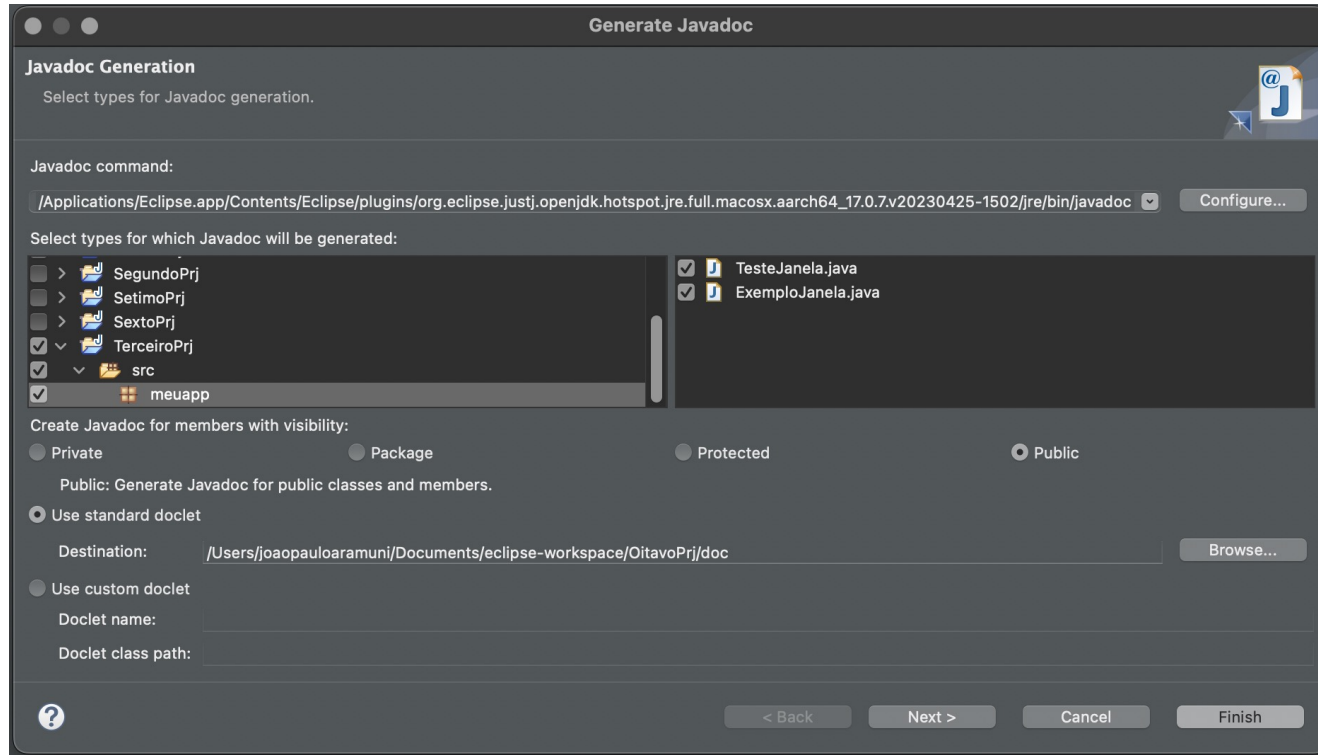
```
ExemploRecursao.java X
6
7  /**
8   * Fatorial
9   *
10  * @param n
11  * @return o fatorial de n (n!)
12  */
13  // Um exemplo simples de recursão.
14  // Esta é uma função recursiva.
15  private static int factR(int n) {
16      int result;
17      if (n == 1) {
18          return 1;
19      }
20      // Executa a chamada recursiva a factR( ).
21      result = factR(n - 1) * n;
22      return result;
23  }
24
25  /**
26  * Máximo Divisor Comum recursivo
27  *
28  * @param a
29  * @param b
30  * @return 0 MDC de a e b
31  */
32  private static int mdc(int a, int b) {
33      if (b == 0) {
34          return a;
35      }
36      return mdc(b, (a % b));
37  }
38
```

- Boas práticas de programação
  - 2) Comentários e documentação
    - Para gerar o Javadoc a partir do Eclipse é muito simples:
    - Na barra de menu, selecione o menu Project, depois a opção "Generate Javadoc..."

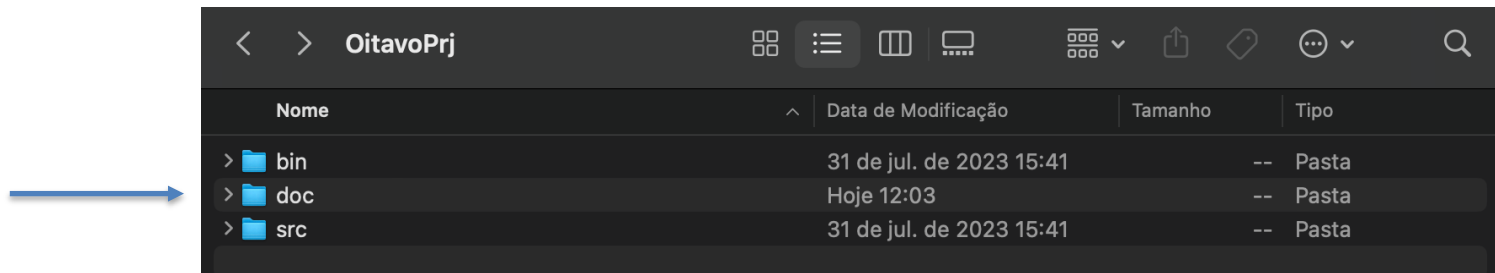


- Boas práticas de programação
  - 2) Comentários e documentação
    - Em seguida, aparecerão as opções para gerar a documentação do seu sistema. Selecione todas as classes.
    - Não esqueça de marcar o caminho da opção "Destination", pois é lá que estará sua documentação.

- 2) Comentários e documentação



- Boas práticas de programação
  - 2) Comentários e documentação
    - Abra a documentação através do caminho que você selecionou.
  - Abra o arquivo **index.html**, que chamará uma página semelhante a essa a seguir.



- 2) Comentários e documentação

MODULE PACKAGE **CLASS** USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD    DETAIL: FIELD | CONSTR | METHOD    SEARCH:

**Module** OitavoPrj  
**Package** meuapp

**Class ExemploRecursao**

java.lang.Object<sup>Ⓔ</sup>  
meuapp.ExemploRecursao

---

```
public class ExemploRecursao
extends ObjectⒺ
```

**Constructor Summary**

**Constructors**

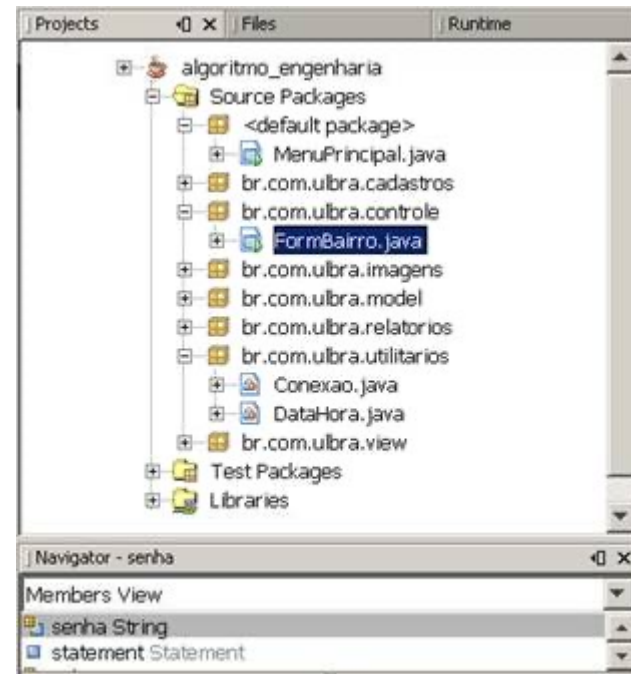
Constructor	Description
<code>ExemploRecursao()</code>	

**Method Summary**

**All Methods**   Static Methods   Concrete Methods

Modifier and Type	Method	Description
static int	<code>fibo(int n)</code>	Números de Fibonacci
static void	<code>main(String<sup>Ⓔ</sup>[] args)</code>	
static double	<code>potencia(int n, int pot)</code>	Potência recursiva

- Boas práticas de programação
  - 3) Pacotes (packages)
    - Uso de pacotes para a separação de códigos em categorias, contribuindo para um projeto elegante e ordenado.
  - Procure separá-las por categorias, funcionalidades ou pela relação que existe entre elas.





- 4) Convenções de nomes para classes, métodos, variáveis e pacotes:
  - **Pacotes:**
    - Durante o desenvolvimento, devemos agrupar as classes de acordo com seus objetivos.
    - Por exemplo, para classes responsáveis pela interface com o usuário, podemos definir o pacote `br.com.ulbra.view`.
      - Observando-o, facilmente identificamos que ele deve ser escrito de forma semelhante a um endereço web, só que de trás para frente. E, ao final, indicamos um nome (ou um conjunto de nomes, preferencialmente separados por “.”), que classifica as classes agrupadas;

- 4) Convenções de nomes para classes, métodos, variáveis e pacotes:
  - **Classes e Interfaces:**
    - Os nomes das classes iniciam com uma letra maiúscula, sendo simples e descritivo.
    - Quando a classe possuir um nome composto, como MenuPrincipal, o primeiro caractere de cada palavra deve ser sempre maiúsculo.
    - Essas regras também são aplicadas para Interfaces;

- 4) Convenções de nomes para classes, métodos, variáveis e pacotes:
  - **Métodos:** O que difere a convenção de nomes de classes para nomes de métodos é que para os métodos a primeira letra deve estar em minúsculo.
  - Como os métodos são criados para executar algum procedimento, procure usar verbos de ação para seus nomes, por exemplo: `imprimirValor()`;

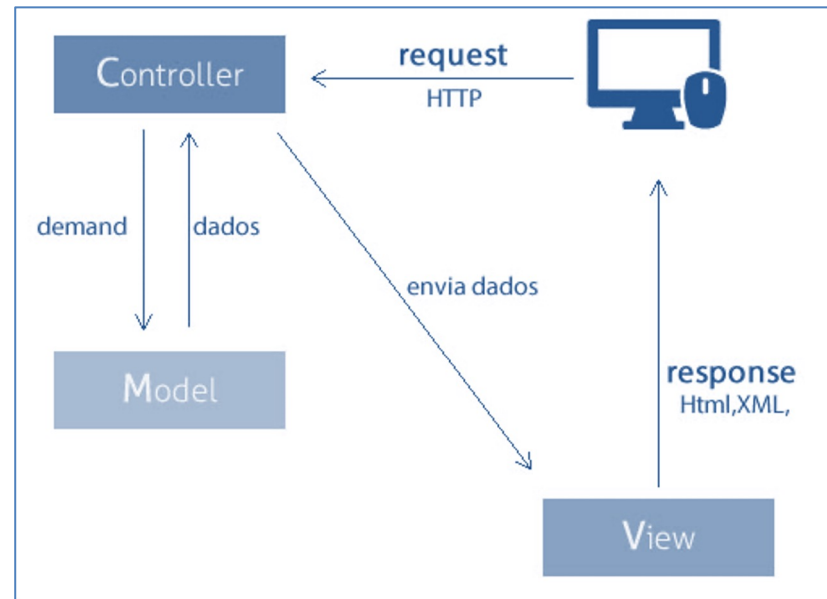
- 4) Convenções de nomes para classes, métodos, variáveis e pacotes:
  - **Variáveis:** Use a mesma convenção adotada para Métodos, criando sempre nomes curtos (nota, primeiroNome, mediaAluno) e significativos.
  - Evite nomes de variáveis de apenas um caractere (i, j, x, y), a não ser para aquelas usadas para índices em laços de repetição.
  - Em variáveis finais (constantes), isto é, que possuem um valor fixo, todas as letras devem estar em maiúsculo e com as palavras separadas por sublinhado (“\_”), por exemplo: TAXA\_IMPOSTO.

- Boas práticas de programação
  - 5) Tratamento de erros (try/catch)
    - Java é considerado uma linguagem robusta, e um dos motivos é o fato dela exigir que você use o try..catch em determinadas situações, como no acesso ao banco de dados ou para a seleção de um arquivo externo qualquer.
    - Isso já garante o uso de uma boa prática. Para outros casos, é altamente recomendável o tratamento de erros, oferecendo assim maior segurança e confiabilidade ao sistema.

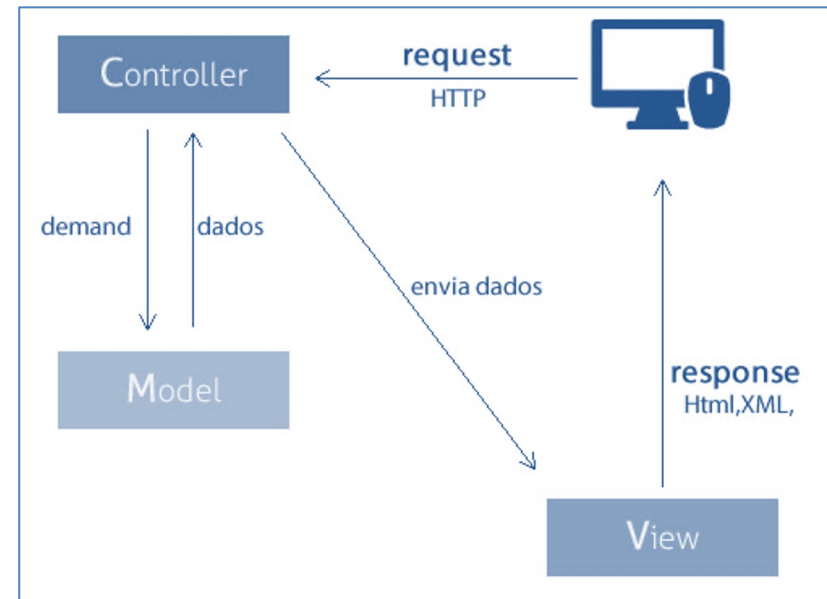
- Boas práticas de programação
  - 5) Tratamento de erros (try/catch)
    - Obrigatório em comunicação com banco de dados.

```
1 public boolean conecta(){
2
3     boolean result = true;
4     try { //tenta conectar
5         Class.forName(driver); //carrega o driver do banco
6         conexao = DriverManager.getConnection(url, usuario, senha);
7         JOptionPane.showMessageDialog(null, "Conexao com sucesso");
8     }
9     catch(ClassNotFoundException erroClass) {
10         JOptionPane.showMessageDialog(null, "Driver não Localizado: "+erroClass);
11         result = false;
12     }
13     catch(SQLException erroBanco){
14         JOptionPane.showMessageDialog(null, "Problemas na comunicação com o banco: "+erroBanco);
15         result = false;
16     }
17
18     return result;
19 }
```

- Boas práticas de programação
  - 6) Padrões de Projetos (Design Patterns)
    - Model-view-controller (MVC) é um padrão de arquitetura de software que divide a aplicação em três camadas: Model, View e Controller.

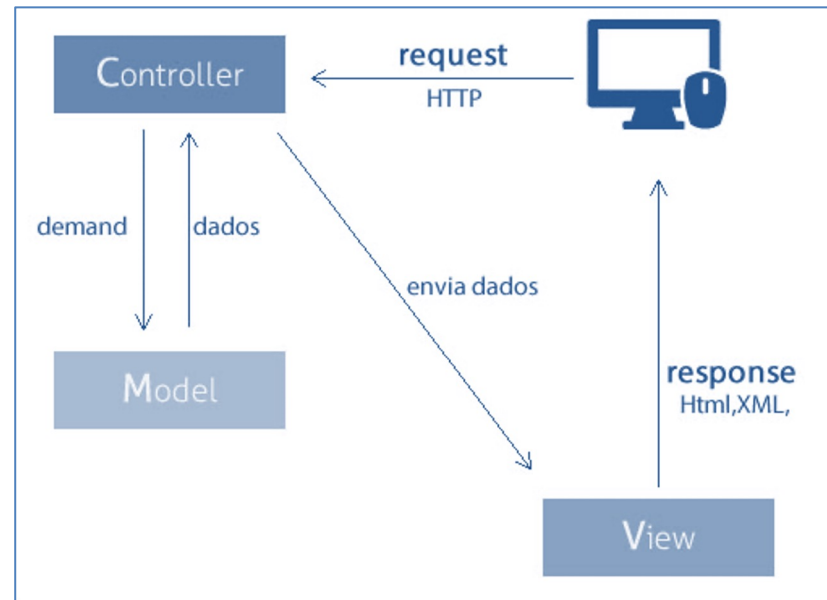


- Boas práticas de programação
  - 6) Padrões de Projetos (Design Patterns)
    - **M (MODEL)**
    - A camada Model (modelo) é responsável pela leitura, escrita e validação dos dados.
    - Nesta camada são implementadas as regras de negócios. Sempre que você pensar em manipulação de dados, pense em model.

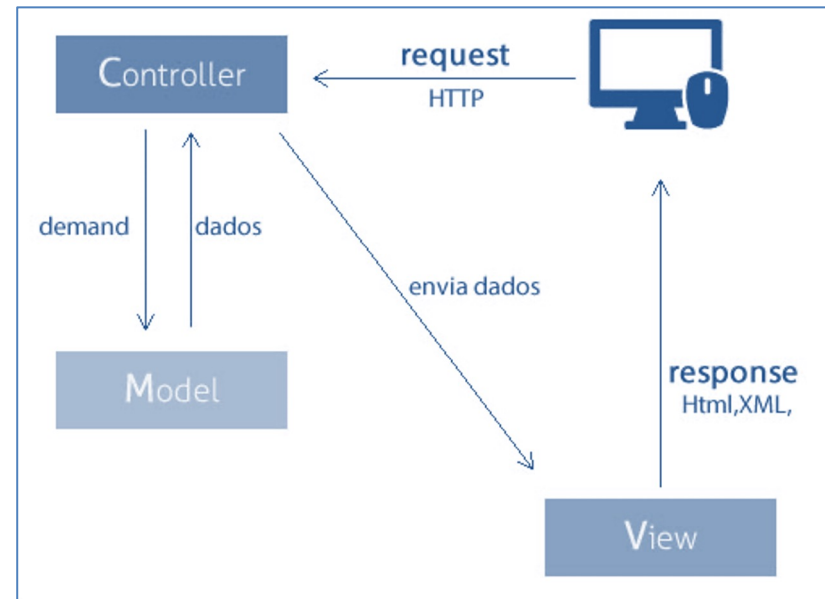




- Boas práticas de programação
  - 6) Padrões de Projetos (Design Patterns)
    - **V (VIEW)**
    - A camada de visão é responsável pela interação com o usuário. Nesta camada são apresentados os dados ao usuário.
    - Os dados podem ser entregues em vários formatos, dependendo do que for preciso, como páginas HTML, arquivos XML, documentos, vídeos, fotos, músicas, entre outros.



- Boas práticas de programação
  - 6) Padrões de Projetos (Design Patterns)
    - **C (CONTROLLER)**
    - A camada Controller (controlador) é responsável por lidar com as requisições do usuário.
    - Ela gerencia as ações realizadas, fala qual Model e qual View utilizar, para que a ação seja completada.




- Boas práticas de programação
  - 7) Deprecation
    - Ao dar manutenção em um código e/ou migrá-lo para uma nova versão do Java, procure analisá-lo e fazer as mudanças necessárias.
    - Para facilitar esta tarefa, as IDEs modernas, como NetBeans e Eclipse, facilitam o nosso trabalho destacando estes elementos no código.
      - Exemplo: método **show()** (**depreciado**), empregado em aplicações desktop para carregar um formulário. Atualmente, para esta mesma funcionalidade, devemos utilizar **setVisible()**.

- Boas práticas de programação
  - 7) Deprecation

```
public class TesteJanela extends JFrame{  
    // Atributos  
    private JPanel painel = new JPanel();  
    private JButton jButtonLimpar = new JButton("Limpar");  
    private JTextField jTextFieldTexto = new JTextField("Teste", 20);  
    private JLabel jLabelMensagem = new JLabel("Exemplo de Simples Janela");  
  
    // Construtor  
    public TesteJanela(){  
        this.setTitle("Exemplo de Interface Gráfica");  
        this.setSize(400,200);  
        configurarComponentes();  
        this.setLocationRelativeTo(null); // Centralizar janela  
        this.setVisible(true); // Exibir janela  
        show();  
    }  
    private void configurarComponentes(){  
        jButtonLimpar.addActionListener(new ActionListener(){  
            @Override public void actionPerformed(ActionEvent e){  
                jTextFieldTexto.setText("");  
                jLabelMensagem.setText("Limpar");  
            }  
        });  
    }  
}
```

The method show() from the type Window is deprecated

1 quick fix available:

-  [Configure problem severity](#)

- Boas práticas de programação
  - 8) Teste e depuração
    - A busca por qualidade levou os testes de software a ganharem um espaço muito importante dentro do ciclo de desenvolvimento de sistemas.
    - Criar testes automatizados se tornou parte da rotina padrão de desenvolvimento.
    - Ao criar casos de teste automatizados, as equipes de desenvolvimento podem identificar rapidamente erros e defeitos, garantindo que o software atenda aos requisitos e funcione conforme o esperado.

- Boas práticas de programação
  - 8) Teste e depuração
    - Além disso, testes automatizados permitem a validação contínua do código, facilitando a detecção precoce de problemas e a implementação de correções rápidas, o que acelera o ciclo de desenvolvimento.
    - Isso resulta em um código mais robusto e seguro, reduzindo a ocorrência de falhas em produção e proporcionando maior confiança tanto para os desenvolvedores quanto para os usuários do software.

- Boas práticas de programação
  - 8) Teste e depuração
    - A automação dos testes também otimiza a eficiência da equipe, liberando tempo para se concentrarem em melhorias e inovações, além de permitir a realização de testes repetitivos e abrangentes, que seriam tediosos ou inviáveis manualmente.
    - Os testes automatizados são fundamentais para um desenvolvimento ágil e confiável, garantindo a entrega de produtos de alta qualidade e que atendam às expectativas dos clientes.

- Boas práticas de programação
  - 8) Teste e depuração
    - Exemplo de teste com JUnit 5

```
1 public class Calculadora {  
2     public int multiplicar(int a, int b) {  
3         return a * b;  
4     }  
5 }
```

```
1 package meuapp;  
2  
3 import org.junit.jupiter.api.Test;  
4 import static org.junit.jupiter.api.Assertions.assertEquals;  
5  
6 public class CalculadoraTest {  
7  
8     @Test  
9     public void testMultiplicar() {  
10         // Cria uma instância da Calculadora  
11         Calculadora calculadora = new Calculadora();  
12  
13         // Executa o método multiplicar com dois valores  
14         int resultado = calculadora.multiplicar(4, 3);  
15  
16         // Verifica se o resultado é o esperado (4 * 3 = 12)  
17         assertEquals(12, resultado);  
18     }  
19 }
```

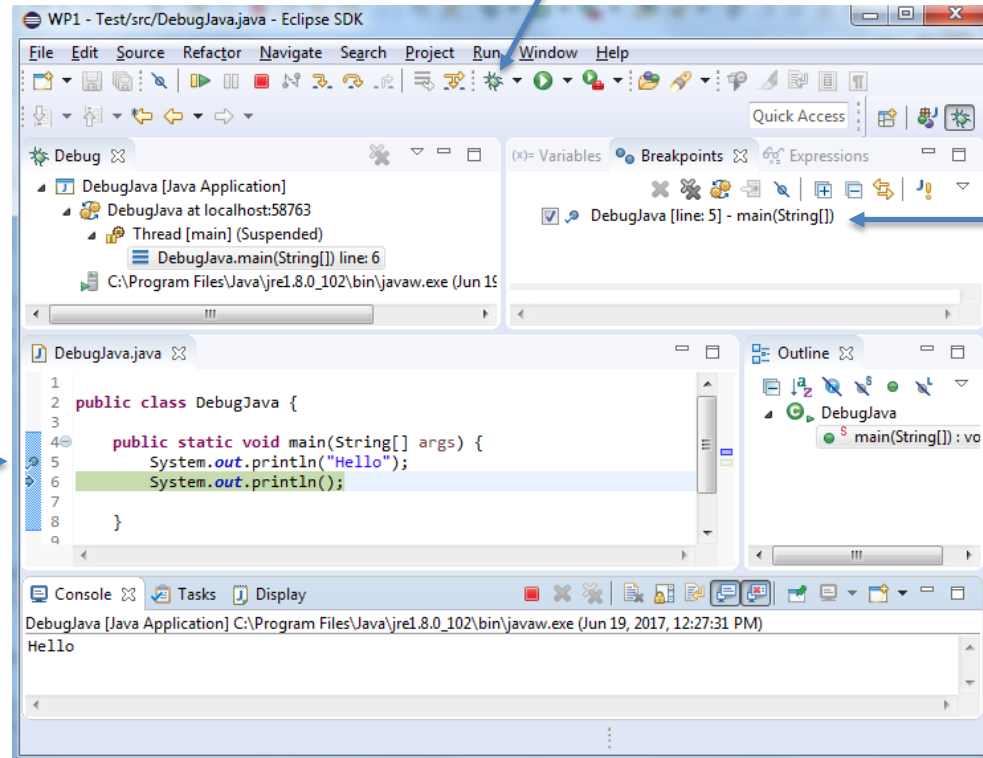


- Boas práticas de programação
  - 8) Teste e depuração
    - Depuração, também conhecida como "debugging" em inglês, é o processo de identificar, analisar e corrigir erros (bugs) em um programa de computador.
    - Esses erros podem incluir comportamentos inesperados, falhas, exceções ou qualquer outro problema que impeça o programa de funcionar corretamente. A depuração é uma etapa essencial no desenvolvimento de software, pois nenhum programa é perfeito desde o início.

- Boas práticas de programação
  - 8) Teste e depuração
    - Para depurar um programa, os desenvolvedores utilizam ferramentas específicas, chamadas de "depuradores", que permitem que eles executem o programa passo a passo, inspecionem variáveis, encontrem a origem dos erros e entendam como o programa está se comportando em tempo de execução.

## 8) Teste e depuração

- Exemplo de debug com Eclipse:



Abrir o debug

Lista de breakpoints

Breakpoint:  
Duplo click na linha

- 8) Teste e depuração
  - Atalhos de debug no Eclipse:
    - Iniciar Depuração: F11
      - Este atalho inicia a execução do programa em modo de depuração.
    - Parar Depuração: Ctrl + F2
      - Esse atalho encerra a execução do programa em modo de depuração.
    - Continuar: F8
      - Esse atalho permite que o programa continue a execução normal após pausas, como pontos de interrupção.
    - Próxima Linha: F6
      - Com esse atalho, você avança para a próxima linha de código durante a depuração.
    - Entrar no Método: F5
      - Esse atalho permite que você entre em um método chamado a partir do ponto atual durante a depuração.

- 8) Teste e depuração
  - Atalhos de debug no Eclipse:
    - Sair do Método: F7
      - Esse atalho permite que você saia de um método em execução durante a depuração e retorne ao método chamador.
    - Exibir Detalhes: Ctrl + Shift + I
      - Esse atalho mostra detalhes sobre uma variável, como o valor atual dela, durante a depuração.
    - Adicionar/Remover Ponto de Interrupção: Ctrl + Shift + B
      - Esse atalho permite adicionar ou remover um ponto de interrupção no código.
    - Exibir Variáveis: Ctrl + Shift + D
      - Esse atalho mostra uma janela com as variáveis disponíveis durante a depuração.
    - Exibir Pilha de Chamadas: Ctrl + 5 (ou Ctrl + F7)
      - Esse atalho exibe a pilha de chamadas (stack trace) atual durante a depuração, mostrando o histórico de chamadas de métodos até o ponto atual.

- Boas práticas de programação
  - 9) Tamanho
    - Evite classes muito longas; com mais de 1000 linhas já fica muito complicado a leitura e depuração do código, por isso devem ser evitadas.
    - Quando uma classe se torna muito extensa, várias questões podem surgir, incluindo:
      - **Complexidade:** Classes longas tendem a ter muitas responsabilidades e funcionalidades, o que pode torná-las mais difíceis de entender e manter.
      - Dificuldade de **manutenção**: Alterar uma classe longa pode ser mais complicado e arriscado, pois qualquer mudança pode afetar várias partes do código.

- Boas práticas de programação
  - 9) Tamanho
    - Evite classes muito longas; com mais de 1000 linhas já fica muito complicado a leitura e depuração do código, por isso devem ser evitadas.
    - Quando uma classe se torna muito extensa, várias questões podem surgir, incluindo:
      - **Reutilização:** Classes grandes podem não ser tão reutilizáveis, pois elas são menos modulares e podem não se encaixar facilmente em outros contextos.
      - **Testabilidade:** Testar classes longas pode ser mais trabalhoso, pois é necessário abordar todos os cenários possíveis para garantir a corretude do código.

- Boas práticas de programação
  - 10) Ferramentas de versionamento
    - Usar ferramentas de versionamento para controle de software e trabalho em equipe, como: TFS (Team Foundation Server), GIT, Mercurial, Subversion, etc.

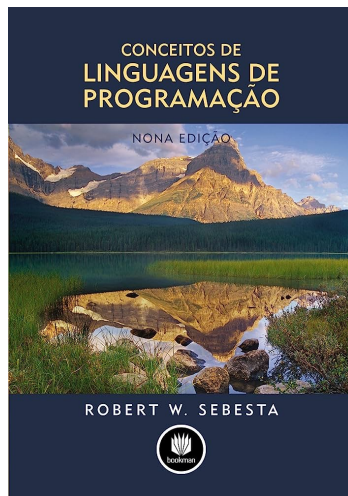
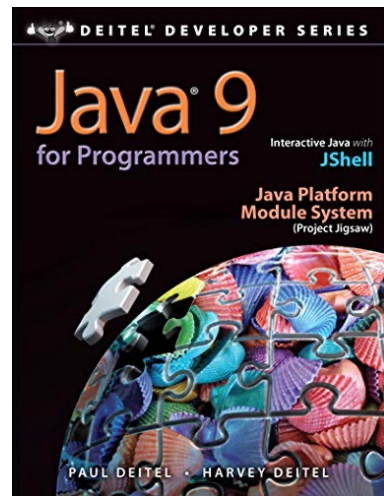


git





# Referências





Obrigado!

joao.aramuni@newtonpaiva.br