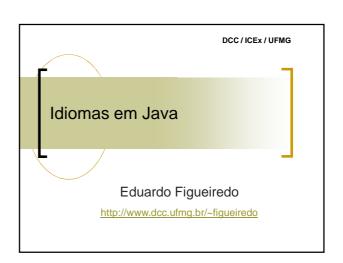


Definição de Idiomas

- Idiomas são padrões de baixo nível específicos de uma linguagem de programação
- Cada idioma descreve como resolver um problema de programação em uma determinada linguagem
- Idiomas facilitam a comunicação entre programadores
 - Aceleram o desenvolvimento
 - o Facilitam atividades de manutenção

Um conjunto de idiomas definem um estilo de programação O estilo de programação é definido pela forma como são usadas as construções da linguagem Exemplos de idiomas A forma como os *loops* são usados O formato de nomes A formatação do código fonte



Idioma e Recomendação

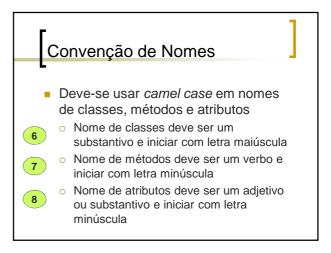
- Muitos dos idiomas são, na verdade, recomendação sobre o uso de construções da linguagem
- Outros idiomas dizem como implementar um padrão de projeto usando uma linguagem
 - Exemplo: como implementar o padrão Adapter em Java

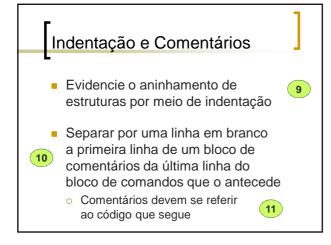
Deve-se declarar uma única classe por arquivo Java A única classe do arquivo deve ser pública para que outras classes tenham acesso Exemplo Arquivo Carro.java

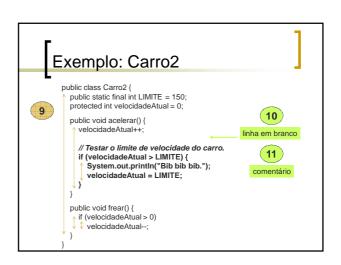


Ocultando Atributos Atributos devem ser privados ou protegidos Métodos get e set devem ser usados por outras classes para acessar os atributos public class Carro { private String cor; protected int velocidadeAtual = 0; public void acelerar() {...} public void setCor(String novaCor) { cor = novaCor; } public String getCor() { return cor; } }

```
Exemplo: Carro
                                           public class CarroTeste {
  public static void main(String[] args) {
public class Carro {
 private String cor;
                                              Carro meuCarro = new Carro();
 protected int velocidadeAtual = 0;
                                              meuCarro.setCor("Preto");
 public void acelerar() {
                                              System.out.println(meuCarro);
   velocidadeAtual++;
 public void frear() {
  velocidadeAtual--
 public void setCor(String novaCor) { cor = novaCor; }
 public String getCor() { return cor; }
 public String toString() {
  return "Carro" + cor + " : " + velocidadeAtual;
                                                  Carro.java
```







Reduzir Escopo

- Utilize blocos aninhados para declarar variáveis locais de modo que tenham o menor escopo possível
 - o Blocos aninhados também são úteis para delimitar a região de um comentário

```
Exemplo: Carro3
   public class Carro3 {
   public static final int LIMITE = 150;
     protected int velocidadeAtual = 0:
     public void acelerar() {
        velocidadeAtual++;
        String alerta = "Bib.";
if (velocidadeAtual == LIMITE)
System.out.println(alerta);
                                                                          bloco aninhado
       // Testar o limite de velocidade do carro. if (velocidadeAtual > LIMITE) {
         velocidadeAtual = LIMITE:
```

Declarações

- Evite nomear variáveis locais com o mesmo nome de variáveis globais (13)
 - Ou com nomes de métodos da classe
 - Sempre que possível, declare e inicialize as variáveis em um mesmo comando

Expressões

- É proibido o uso de goto
- Evite o uso de operador ternário "?" quando uma das expressões contiver mais de um operador
 - o Neste caso, use o comando if

```
public void fear() {
  velocidadeAtual = ( velocidadeAtual < 0) ?</pre>
        ( velocidadeAtual == MIN ? STOPPED :
velocidadeAtual - min() ) :
velocidadeAtual - desacelerar();
```

Switch Case

- Mantenha curto o código de cada case de um switch
 - o Em torno de 5 linhas
- **17**
- Código longo deve ser extraído para um método
- Sempre termine o case com um comando break

Default de um Switch

- Sempre inclua uma opção default 19 nas estruturas switch
- O default deve capturar somente as condições não previstas pelos case 20

Repetições

- Não crie variáveis temporárias apenas para término de uma repetição
 - Use o comando break para sair de um laço de repetição antes da condição de saída ser atingida
 - Use o comando return para testar imediatamente a condição de saída

Expressões

- Evite expressões lógicas complexas como condição de um if
 - Particione-as em vários comandos if aninhados
- Todos os blocos { } vazios devem receber um comentário indicando que estão propositalmente vazios

Bibliografia

- F. Buschmann et al. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 1996.
 - o Cap. 4 Idioms
- A. von Staa. Programação Modular. Elsevier, 2000.
 - o Apêndices 3, 4 e 5