

Idiomas de Programação

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

[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
 - Facilitam atividades de manutenção

[Idiomas e Estilos]

- 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



Idiomas em Java

Eduardo Figueiredo

<http://www.dcc.ufmg.br/~figueiredo>

[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

Uma Classe por Arquivo

- Deve-se declarar uma única classe por arquivo Java 1
 - A única classe do arquivo deve ser pública para que outras classes tenham acesso
- Exemplo

Arquivo Carro.java



```
public class Carro {  
    String cor;  
    int velocidadeAtual;  
  
    void acelerar() {}  
    void frear() {}  
}
```

[O Método Main]

- Deve-se colocar o método *main()* em uma classe separada 2
- Apenas código de iniciação do sistema deve estar na classe que contém o método *main()*
- Exemplo

```
public class TesteCarro {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

Ocultando Atributos

4

- Atributos devem ser privados ou protegidos

5

- 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 frear() {...}

    public void setCor(String novaCor) { cor = novaCor; }
    public String getCor() { return cor; }
}
```

[Exemplo: Carro]

```
public class Carro {
    private String cor;
    protected int velocidadeAtual = 0;

    public void acelerar() {
        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

```
public class CarroTeste {
    public static void main(String[] args) {
        Carro meuCarro = new Carro();
        meuCarro.setCor("Preto");
        System.out.println(meuCarro);
    }
}
```

CarroTeste.java

[Convenção de Nomes]

- Deve-se usar *camel case* em nomes de classes, métodos e atributos

6

- Nome de classes deve ser um substantivo e iniciar com letra maiúscula

7

- Nome de métodos deve ser um verbo e iniciar com letra minúscula

8

- Nome de atributos deve ser um adjetivo ou substantivo e iniciar com letra minúscula

[Indentação e Comentários]

- Evidencie o aninhamento de estruturas por meio de indentação

9

- Separar por uma linha em branco a primeira linha de um bloco de comentários da última linha do bloco de comandos que o antecede

10

- Comentários devem se referir ao código que segue

11

[Exemplo: Carro2]

9

```
public class Carro2 {  
    public static final int LIMITE = 150;  
    protected int velocidadeAtual = 0;  
  
    public void acelerar() {  
        velocidadeAtual++;  
  
        // Testar o limite de velocidade do carro.  
        if (velocidadeAtual > LIMITE) {  
            System.out.println("Bib bib bib.");  
            velocidadeAtual = LIMITE;  
        }  
    }  
  
    public void frear() {  
        if (velocidadeAtual > 0)  
            velocidadeAtual--;  
    }  
}
```

10

linha em branco

11

comentário

[Reduzir Escopo]

- Utilize blocos aninhados para declarar variáveis locais de modo que tenham o menor escopo possível
- 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);
        }

        // Testar o limite de velocidade do carro.
        if (velocidadeAtual > LIMITE) {
            velocidadeAtual = LIMITE;
        }
    }
    ...
}
```

12

bloco aninhado

[Declarações]

13

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

14

[Expressões]

- Evite o uso de operador ternário “?” quando uma das expressões contiver mais de um operador
 - Neste caso, use o comando *if*

15

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

[*Switch Case*]

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

16

17

[*Default* de um *Switch*]

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

18

19

[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

21

22

[Bibliografia]

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