DCC / ICEx / UFMG

Programming Idioms

Eduardo Figueiredo

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

Programming Idioms

- Idioms are low-level patterns specific to a programming language
- An idiom describes how to solve a implementation-specific problem in a programming language
- Idioms ease communication among developers
  - Also speed up development and maintenance tasks

Idioms and Style

- A collection of related idioms defines a programming style
  - A programming style is characterized by the way language constructs are used
- Examples of style
  - The kind of loop statement used
  - Naming of program elements
  - Formatting of the source code

DCC / ICEx / UFMG

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 dizem como implementar um padrão de projeto usando os mecanismos de uma linguagem
  - Exemplo: como implementar o Singleton em Java

Uma Classe por Arquivo

- 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

```
public class Carro {
  String cor;
  int velocidadeAtual;

  void acelerar() {}
  void frear() {}
}
```

O Método Main

- Deve-se colocar o método *main()* em uma classe separada
  - 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

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

Convenção de Nomes

- Deve-se usar *camel case* em nomes de classes, métodos e atributos
  - Nome de classes deve ser um substantivo e iniciar com letra maiúscula
  - Nome de métodos deve ser um verbo e iniciar com letra minúscula
  - Nome de atributos deve ser um adjetivo ou substantivo e iniciar com letra minúscula

Endentação e Comentários

- Evidencie o aninhamento de estruturas por meio de endentação
- Separa por uma linha em branco a primeira linha de um bloco de comentários da última linha do bloco de comandos que o antecede
  - Comentários devem se referir ao código que segue

Reduzir Escopo

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

```
public void acelerar() {
  velocidadeAtual = velocidadeAtual + 1;
  {
    String message = getMessage();
    System.out.println(message);
  }
  ...
}
```

Declarações

- 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

Expressões

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

```
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 trono de 5 linhas
  - 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* nas estruturas *switch*
- O *default* deve capturar somente as condições não previstas pelos *case*

Repetições

- Não crie variáveis temporárias apenas para término de uma repetição
  - Use os comando *break* e *return* para sair de um laço de repetição antes da condição de saída ser atingida

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.
  - Cap. 4 Idioms
- A. von Staa. Programação Modular. Elsevier, 2000.
  - Apêndices 3, 4 e 5