Encapsulamento, membros estáticos e finais

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Encapsulamento: ocultando informações

- Objetiva separar aspectos visíveis de um objeto ou classe de seus detalhes de implementação
- Interface:
  - tudo aquilo que o usuário do objeto vê/acessa.
- Permite que seus dados sejam protegidos de acesso ilegal.

Modificadores de acesso

O Java possui 4 modificadores de acesso ao nível dos membros:

- **private**: membros declarados com acesso privado são acessíveis apenas na própria classe.
- *package-private*: membros declarados sem modificador de acesso são acessíveis apenas às classes dentro do mesmo pacote.
- **protected**: membros declarados com acesso protegido são acessíveis às classes do pacote e adicionalmente por suas subclasses.
- **public**: membros declarados com acesso público são acessíveis de qualquer lugar do programa.

Princípios da ocultação de informação

- Use o nível de acesso mais restrito e que faça sentido para um membro particular.
- Use **private** a menos que haja uma boa razão para não fazê-lo.
- Evite campos **public** exceto para constantes. Campos públicos aumentam o acoplamento em relação a uma implementação específica e reduz a flexibilidade do sistema a mudanças.

Classe Produto: encapsulamento

```
public class Produto {
    private String descricao;
    private float preco;
    private int quant;

    public bool emEstoque() {
        return (quant > 0);
    }

    public Produto(String d, float p, int q)  {
        ...
    }

    public Produto() {
        ...
    }
}
```

Métodos de acesso (*getters* e *setters*)

- Métodos *get*: acessam o valor de um atributo privado.
  - Valores podem ser tratados antes de serem exibidos.
  - Ex: atributo booleano sendo exibido como V ou F atributo numérico e seu correspondente string.

  Métodos *set*: atribuem um valor a um atributo privado.
  - Valores devem ser validados/tratados antes de serem atribuídos.
  - Ex: número do `dia` numa classe Data depende do atributo `mes`.

Classe Produto: métodos de acesso (*getters* e *setters*)

```
public String getDescricao() { return descricao; }
public float getPreco() { return preco; }
public int getQuant() { return quant; }

public void setDescricao(String d) {
    if (d.length() >= 3)    descricao = d;
}
public void setPreco(float p) {
    if (preco > 0)   preco = p;
}
public void setQuant(int q) {
    if (quant >= 0)   quant = q;
}
public Produto(String d, float p, int q)
{
    setDescricao(d);
    setPreco(p);
    setQuant(q);
}
```

Classe Produto: acessando membros encapsulados

```
class Aplicacao {
    public static void main(String args[])
    {
        Produto p1 = new Produto();
        Produto p2 = new Produto("Shulambs,1.99F,200);

        p1.setDescricao("Cool Shulambs");
        p1.setPreco(2.49F);
        p1.setQuant(10);

        System.out.println("Produto: " + p1.getDescricao());
        System.out.println("Preço: " + p1.getPreco());
        System.out.println(" Estoque: " + p1.getQuant());

        System.out.println("Produto: " + p2.getDescricao());
        System.out.println("Preço: " + p2.getPreco());
        System.out.println(" Estoque: " + p2.getQuant());

    }
}
```

Quando não utilizar métodos de acesso

```
class Conta {
    private double limite;
    private double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public double getLimite() {
        return limite;
    }

    public void setLimite(double limite) {
        this.limite = limite;
    }
}
```

```
class Conta {
    private double saldo;
    private double limite;

    public Conta(double limite) {
        this.limite = limite;
    }

    public void depositar(double x) {
        this.saldo += x;
    }

    public void sacar(double x) {
        if(this.saldo + this.limite >= x) {
        this.saldo -= x;
        }
        else throw
          new Exception("Fundos insuficientes.");
        }

    public double getSaldo() {
        return this.saldo;
    }
}
```

Membros estáticos

Membro estático

Membro com tempo de vida global e escopo local.

- São atributos ou métodos comuns a todos os objetos de uma classe.
- Membros de classe.
  - Compartilhado por todos os objetos daquela classe.
  - Primeiro objeto inicializa a variável[1].

[1] Na verdade, em Java, membro é inicializado quando a classe é carregada.

Membros estáticos

- Úteis para implementar contadores ou identificadores de autoincremento.
- Podem ser usados para definir constantes.
  - Como a variável e compartilhada por todos os objetos de uma classe, a utilização de membros estáticos constantes pode permitir grande economia de memória.
- Modificador de acesso **static**.

Exemplo: Identificador estático de Produtos

```
class Produto {
    ...
    private int id;
    private static int cont = 0;

    public static int getCont() {
        return cont;
    }
    public int getId() {
        return id;
    }
    ...
    public Produto(String d, float p, int q) {
        setDescricao(d);
        setPreco(p);
        setQuant(q);

        id = ++cont;
    }
    public Produto() {
        descricao = "Novo Produto";
        preco = 0.01F;
        quant = 0;

        id = ++cont;
    }
}
```

Exemplo: Identificador estático de Produtos

```
class Aplicacao {
    public static void main(String args[]) {

        System.out.println("\nCont. prods: " + Produto.getCont());

        Produto p1 = new Produto();
        System.out.println("\nCont. prods: " + Produto.getCont());

        Produto p2 = new Produto("Shulambs", 1.99F, 200);
        System.out.println("\nCont. prods: " + Produto.getCont());
        ...

        System.out.println("Produto id: " + p1.getId());
        System.out.println("Descrição:  " + p1.getDescricao());
        System.out.println("Preço:      " + p1.getPreco());
        System.out.println("Estoque:    " + p1.getQuant());

        System.out.println("Produto id: " + p2.getId());
        System.out.println("Descrição:  " + p2.getDescricao());
        System.out.println("Preço:      " + p2.getPreco());
        System.out.println("Estoque:    " + p2.getQuant());

    }
}
```

Membros finais

Membro final

Podem ser definidos/inicializados apenas uma vez.

- São atributos, métodos ou classes, em geral, constantes.
- Modificador de acesso **final**.
- Sua função varia dependendo do tipo de membro. No caso de variáveis, define constantes.

Classe Produto: atributo final

```
class Produto {
    public static final String DESCRICAO_PADRAO = "Shulambs";
    public static final int MAX_ESTOQUE = 1000;
    ...

    public void setQuant(int q) {
        if (q >= 0 && q <= MAX_ESTOQUE)
            quant = q;
    }
    ...

    public Produto() {
        descricao = DESCRICAO_PADRAO;
        preco = 0.01F;
        quant = 0;

        id = ++cont;
    }
}
```

Classe Produto: atributo final

```
class Aplicacao {
   public static void main(String args[]) {

      Produto p1 = new Produto();
      Produto p2 = new Produto("Shulambs", 1.99F, 600);

      int novosProdutos = Integer.parseInt(
            JOptionPane.showInputDialog(null,
            "Digite quantos produtos deseja adicionar ao estoque:",
            "Controle de estoque",
            JOptionPane.OK_CANCEL_OPTION));

      if ((p2.getQuant() + novosProdutos) > Produto.MAX_ESTOQUE) {
         JOptionPane.showMessageDialog(null,
               "Estourou o limite máximo do estoque.",
               "Erro alterando estoque", JOptionPane.ERROR_MESSAGE);
      } else {
         p2.setQuant(p2.getQuant() + novosProdutos);
      }
}
```