

Encapsulamento

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC

Encapsulamento - Idéia

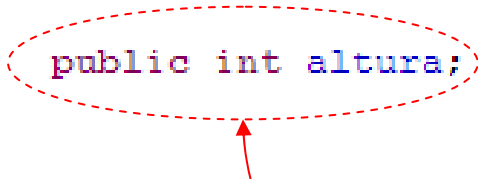


Este gato representa um objeto da classe Gato, e possui altura como característica. Todo e qualquer gato não pode possuir altura inferior a 9.

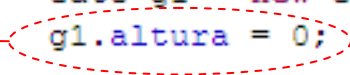
O que pode acontecer se não usarmos encapsulamento?

Encapsulamento - Idéia

```
3 public class Gato {  
4  
5     public int altura;  
6  
7 }
```



```
3 public class Principal {  
4  
5     public static void main(String args[]){  
6  
7         Gato g1 = new Gato();  
8         g1.altura = 0;  
9     }  
10 }
```



Péssimo. Não deve-se deixar isso acontecer.
Como evitar?

Que implicações isso gera?

Encapsulamento - Idéia

```
3 public class Gato {  
4  
5     public int altura;  
6  
7 }
```

Mudar para



```
3 public class Gato {  
4  
5     private int altura;  
6  
7     public void setAltura(int altura){  
8         if(altura >= 9){  
9             this.altura = altura;  
10        }  
11    }  
12 }
```



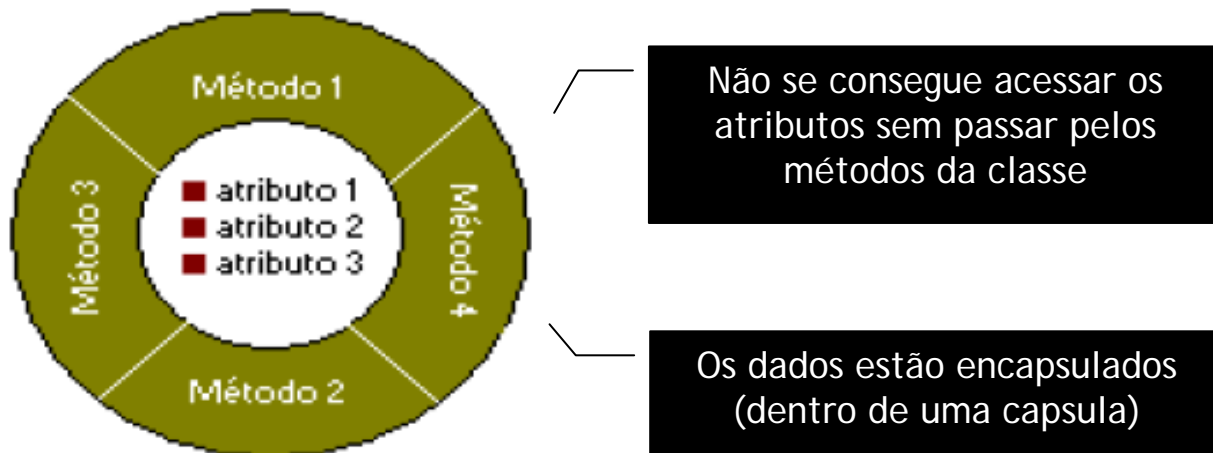
```
3 public class Principal {  
4  
5     public static void main(String args[]){  
6  
7         Gato g1 = new Gato();  
8         g1.setAltura(8);  
9     }  
10 }
```

Óbvio que a alteração para 8 não será executada. Existe um método para impedir isto.

Encapsulamento

■ Definição

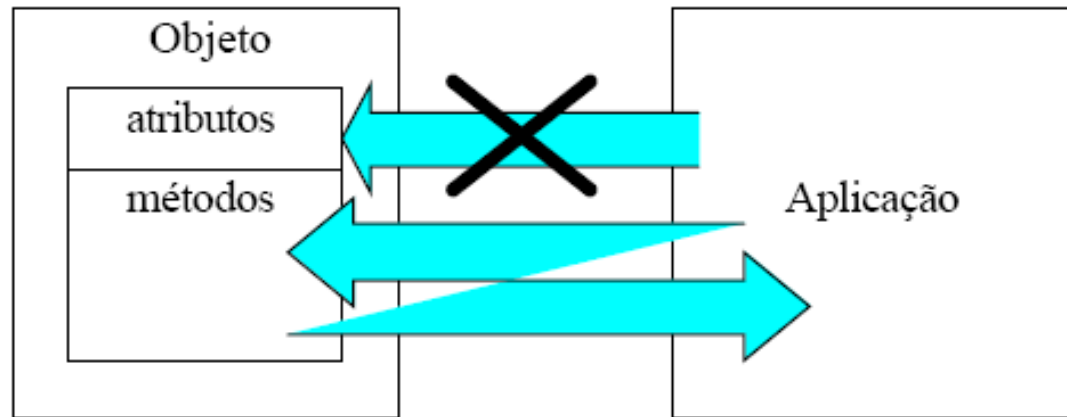
- Empacotamento dos atributos e métodos numa mesma classe. Isto protege os dados contra corrupção, pois somente os métodos da classe poderão alterar as estruturas de dados da classe.



Encapsulamento

■ Definição

- O conceito de encapsulamento é decorrente do fato de se combinar os dados (atributos) e o código que manipula estes dados (métodos) em um único Objeto. Ele garante que a única forma de acesso aos dados é através dos métodos disponíveis ao usuário (chamados públicos). Os demais métodos e os atributos da classe ficam sendo privados.



Encapsulamento

- Definição

- Permite que certas características ou propriedades dos objetos possam ou não ser vistas ou modificadas externamente. A idéia básica relacionada ao encapsulamento é o de **ocultamento de informações**.

Encapsulamento

■ Vantagens

- Esconder a complexidade do Código
 - Esconder detalhes de implementação
- Proteção dos dados
 - São acessados somente via métodos da classe
 - Evita-se que sejam corrompidos por aplicações externas
- Pode-se alterar a implementação do objeto sem alterar a do cliente
 - Objeto a: oferece serviços a outros objetos
 - Objeto b: utiliza serviços oferecidos por a
 - Objeto b é dito ser cliente de a.
 - Pode-se alterar a estrutura interna de "a" sem alterar a estrutura de "b"

Encapsulamento

■ Observação

- Encapsulamento garante que a minha classe seja uma **caixinha preta** para o usuário: ele não sabe o que há dentro do objeto, sabe apenas para que ele serve e quais os métodos disponíveis para a manipulação deste.
- Contudo...
 - Se, ao projetarmos uma classe, não fornecemos métodos de acesso adequados, teremos dificuldades em criar aplicações eficientes com objetos instanciados da mesma.

Exemplificação

```
class Funcionario {  
    double salario;  
}
```

O atributo SALARIO de um objeto da classe FUNCIONARIO pode ser acessado ou modificado por código escrito em qualquer classe ...

- a) Do mesmo pacote da Classe Funcionário
- b) Externa à Classe Funcionário

Exemplificação

```
class Funcionario {  
    private double salario;  
  
    void aumentaSalario(double aumento) {  
        // lógica para aumentar o salário  
    }  
}
```

Incorporando encapsulamento

A variável "salário" passa a ser privada com a declaração `private`. Isso faz com que somente métodos internos dessa classe possam manipular seus valores

Se algum código fora da classe `Funcionario` tentar acessar ou alterar o valor do atributo privado `Salario` de um objeto dessa Classe, um erro de compilação é gerado

Exemplificação

Classe Teste

```
1 public class Teste {
2
3     public static void main(String args[]) {
4
5         Funcionario func = new Funcionario();
6
7         func.salario = 100;
8         System.out.println(func.salario);
9     }
10 }
11
```

Problems Javadoc Declaration Console Progress

<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\jav
100.0

```
1 abstract public class Funcionario {
2
3     public double salario;
4
5     public void aumentarSalario(int aumento) {
6         this.salario = salario + (salario * aumento);
7     }
8
9 }
```

Classe Salário

Exemplificação

Classe Teste

```
1 public class Teste {
2
3     public static void main(String args[]){
4
5         Funcionario func = new Funcionario();
6
7         func.salarario = 100;
8         System.out.println(func.salarario);
9     }
10 }
```

Problems | @ Javadoc | Declaration | Console | Progress

<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\javaw.exe (28/03/2011 09:41:08)

Exception in thread "main" java.lang.Error: Unresolved compilation problems:
The field Funcionario.salarario is not visible
The field Funcionario.salarario is not visible

at Teste.main(Teste.java:7)

```
1 abstract public class Funcionario {
2
3     private double salarario;
4
5     public void aumentarSalario(int aumento){
6         this.salarario = salarario + (salarario * aumento);
7     }
8 }
```

Classe Salário

Exemplificação

Quando salario é "private"

```
1 public class Teste {  
2  
3 public static void main(String args[]) {  
4  
5     Funcionario func = new Funcionario();  
6  
7     func.  
8
```

```
1 public class Teste {  
2  
3 public static void main(String args[]) {  
4  
5     Funcionario func = new Funcionario();  
6  
7     func.  
8  
9  
10 }  
11 }
```

- salario : double - Funcionario
- aumentarSalario(double aumento) : void - Funcionario
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

- aumentarSalario(double aumento) : void - Funcionario
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Quando salario é "public"

Encapsulando

```
1 abstract public class Funcionario {  
2  
3     private double salario;  
4  
5     public void aumentarSalario(int aumento){  
6         this.salario = salario + (salario * aumento);  
7     }  
8  
9     public double imprimir(){  
10         return this.salario;  
11     }  
12 }
```

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         Funcionario func = new Funcionario();  
6  
7         System.out.println(func.imprimir());  
8         func.aumentarSalario(50);  
9         System.out.println(func.imprimir());  
10     }  
11 }
```

Problems @ Javadoc Declaration Console Progress

<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\java
100.0
150.0

Encapsulando

Problema: não há métodos públicos na Classe Funcionário - não existe interface

```
1 public class Teste {  
2  
3     public static void main(String args[]) {  
4  
5         Funcionario func = new Funcionario();  
6  
7         func.  
8     }  
9 }  
10
```

- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

```
1 abstract public class Funcionario {  
2  
3     private double salario;  
4  
5     private void aumentarSalario(int aumento) {  
6         this.salario = salario + (salario * aumento);  
7     }  
8  
9     private double imprimir() {  
10         return this.salario;  
11     }  
12 }
```


Encapsulamento

- Reforçando
 - Definir todos os atributos como privados e definir métodos para implementar as lógicas de acesso e alteração é quase uma regra da orientação a objetos. O intuito é ter sempre um controle centralizado do dados dos objetos para facilitar a manutenção do sistema.

Atributos Privados

■ Acesso

- Para permitir o acesso aos atributos privados de uma maneira controlada, a prática mais comum é criar dois métodos, um que retorna o valor e outro que muda o valor.
- A convenção para esses métodos é de colocar a palavra get ou set antes do nome do atributo.

Atributos Privados

Os métodos com "get" retornam o valor de um atributo, enquanto que aqueles com "set" realizam uma modificação em seu valor

```
public class Conta {  
  
    private double saldo;  
    private double limite;  
    private Cliente titular;  
  
    public double getSaldo() {  
        return this.saldo;  
    }  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public double getLimite() {  
        return this.limite;  
    }  
  
    public void setLimite(double limite) {  
        this.limite = limite;  
    }  
}
```

Uso de Get e Set

- Observação
 - É uma má prática criar uma classe e, logo em seguida, criar getters e setters para todos seus atributos. Você só deve criar um get ou set se tiver a real necessidade.

Exercício

1 – Antes do racionamento de energia ser decretado, quase ninguém falava em quilowatts; mas, agora, todos incorporam essa palavra em seu vocabulário. Sabendo-se que 100 quilowatts de energia custam um sétimo do salário mínimo, fazer uma classe em java que:

- Tenha dois atributos: um que represente o valor do salário mínimo e outro que represente a quantidade de quilowatts gasta por uma residência;
- Encapsule esses atributos;
- Crie um método que retorne o valor em reais de cada quilowatt;
- Crie um método que retorne o valor em reais que a residência terá que pagar;
- Crie um método que retorne o valor em reais que a residência terá que pagar com desconto de 10%;
- Crie um método main que:
 - Atribua um valor aos atributos da classe;
 - Mostre na tela a quantidade em reais que a residência vai pagar, com e sem o desconto.

Encapsulamento

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC