Encapsulamento

Profa. Karen Selbach Borges

Última Atualização: Abr/2021

Definição

- É o processo de *ocultação* das características internas do objeto;
  - Cada parte do problema possui implementação própria e deve realizar seu trabalho independentemente das outras partes.
  - O encapsulamento mantém essa independência, ocultando os detalhes internos através de uma interface externa.

Exemplo

Ao comprar café, você precisa saber como ele é feito pela máquina ?

Exemplo

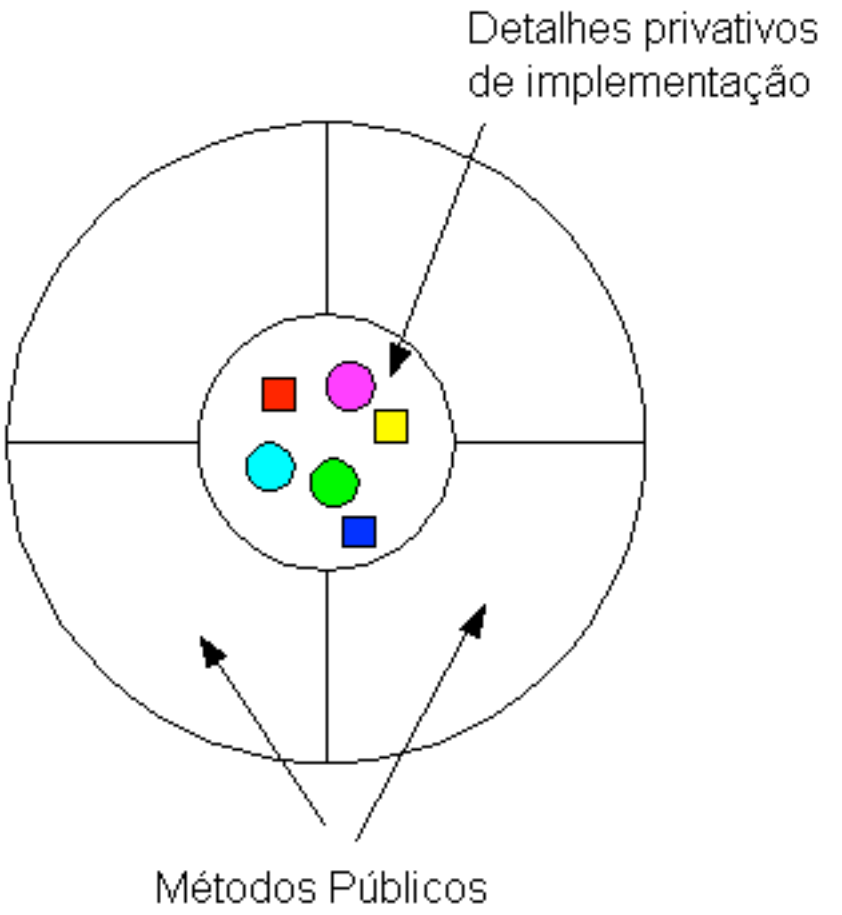
Não !
Os mecanismos de funcionamento estão encapsulados !

Exemplo

A interação se dá através de uma interface pública.

Interface Pública

- Os atributos não podem ser manipulados diretamente.
- Os atributos somente podem ser alterados ou consultados através dos métodos (públicos) do objeto



Porque encapsular?

- Quando usado cuidadosamente, o encapsulamento transforma os objetos em componentes plugáveis.

  - Para que outro objeto utilize este componente, ele só precisa conhecer sua interface pública.

Mecanismos de Encapsulamento

- Métodos
  - Construtores
  - Getters e setters
- Modificadores de visibilidade
- Pacotes de classes

Métodos

- Os métodos formam uma “cerca” em torno dos atributos.
- Os atributos não podem ser manipulados diretamente.
- Os métodos *get* e *set* permitem, respectivamente, obter e atribuir valores às variáveis de instância.

Getters

- Os métodos **get** servem para recuperar o valor de uma variável de instância ou de classe.

```
public class Cliente {
    private String nome;
    private String cpf;

    public Cliente(String nome, String cpf)  {
        this.nome = nome;
        this.cpf = cpf;
    }

    public String getNome(){
       return nome;
    }

    public String getCpf(){
       return cpf;
    }
}
```

Setters

- Os métodos **set** servem para alterar o valor de uma variável de instância ou de classe.

```
public class Cliente {
    private String nome;
    private String cpf;

    public Cliente(String nome, String cpf)  {
        this.nome = nome;
        this.cpf = cpf;
    }

    public void setNome(){
       this.nome = nome;
    }

}
```

Getters e Setters

- Observações:
  - Para classes que representam entidades, é uma boa prática de programação criar getters e setters para cada uma das variáveis de instância.
  - Isso é especialmente útil na implementação de sistemas que usam banco de dados.

Modificadores de Visibilidade

- Definem se o acesso aos atributos e métodos é público, privado ou protegido.

Modificadores de Visibilidades

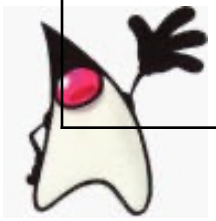
- Public
  - Atributos podem ser acessados e modificados a partir de qualquer classe
  - Pode ser aplicado sobre classes, variáveis, métodos e construtores
- Private
  - Atributos só podem ser acessados e modificados a partir de métodos da própria classe a que pertencem (getters e setters)
  - Pode ser aplicado sobre variáveis, métodos e construtores (situações bem específicas → padrões de projeto. Ex: singleton)

Pacotes de Classes

- Um pacote é um conjunto de classes relacionadas.
- Por exemplo, a biblioteca java (API) é composta por vários pacotes, que reúnem classes afins.

Pacotes de Classes

Pacote	Finalidade	Classes
java.lang	pacote default. Contém classes para manipulação de Srings, funções matemáticas, entre outros.	Object String Math System
java.util	pacote que provê uma miscelânea de classes úteis para manipular estruturas de dados, data, hora, números randômicos, etc.	Arrays Date Random
javax.swing	pacote que provê um conjunto de classes para manipulação de interfaces para o usuário.	JFrame JButton JLabel JTextField



Pacotes de Classes

- No código Java são criados usando a palavra reservada *package*.

a instrução package deve ser declarada na primeira linha de código.

```
package edu.ifrs.exemplo1.classe1;

public class Foo {
   private int x;

   /**
    * Constructor for objects of class Foo
   */
    public Foo()  {
            x = 0;
    }
}
```

Pacotes de Classes

- Uma classe criada dentro de um pacote é utilizada em pacotes diferentes do seu de origem através da palavra reservada *import*.

as instruções de import
são declaradas logo após
o package (se existir);

para usar várias classes de um
mesmo pacote é possível usar *;

```
package edu.ifrs.exemplo1.classe2;
import edu.ifrs.exemplo1.classe1.*;

public class Boo {
   private int z;

public Boo()  {
            z = 0;
            Foo f = new Foo();
            Too t = new Too();
    }
}
```

Pacotes de Classes e
Modificadores de Acesso

- Acesso padrão (ou acesso ao nível de pacote) :
  - Não apresenta um modificador antes da declaração da classe.

    Ex: class Ponto
  - Se a classe A e B estiverem em pacotes diferentes e a classe A tiver acesso padrão, a classe B não poderá criar uma instância de A.

Exemplo 1

```
package edu.ifrs.exemplo1.classe1;

class Foo {
   private int x;

   /**
    * Constructor for objects of class Foo
   */
    public Foo()  {
            x = 0;
    }
}
```

```
package edu.ifrs.exemplo1.classe1;

public class Too {
   private int y;

   /**
    * Constructor for objects of class Too
   */
    public Too()  {
            y = 0;
            Foo f = new Foo();
    }
}
```

Tudo certo aqui porque Foo e Too estão dentro do mesmo pacote

Exemplo 2

```
package edu.ifrs.exemplo1.classe1;

class Foo
{ …
}
```

```
package edu.ifrs.exemplo1.classe1;

public class Too
{ …
}
```

```
package edu.ifrs.exemplo1.classe2;

import edu.ifrs.exemplo1.classe1.*;

public class Boo {
   private int z;

   /**
    * Constructor for objects of class Boo
   */
    public Boo()  {
              z = 0;
              Foo f = new Foo(); // Erro
              Too t = new Too(); // Ok
       }
}
```

Erro aqui porque Foo e Boo estão em pacotes diferentes

Pacotes de Classes e
Modificadores de Acesso

- A regra do acesso padrão também vale para as variáveis:
  - Se a classe A e B estiverem em pacotes diferentes, a classe B não terá acesso às variáveis de A com acesso padrão.

Exemplo 1

```
package edu.ifrs.exemplo1.classe1;

public class Foo {
    int x;

   /**
    * Constructor for objects of class Foo
   */
    public Foo()  {
            x = 0;
     }
}
```

```
package edu.ifrs.exemplo1.classe1;

public class Too {
    int y;

   /**
    * Constructor for objects of class Too
   */
    public Too()  {
          Foo f = new Foo();
          f.x = 1;
     }
}
```

Tudo certo aqui porque Foo e Too estão dentro do mesmo pacote

Exemplo 2

```
package edu.ifrs.exemplo1.classe1;

public class Foo
{ …
}
```

```
package edu.ifrs.exemplo1.classe2;

import edu.ifrs.exemplo1.classe1.*;

public class Boo {
   private int z;

   /**
    * Constructor for objects of class Boo
   */
    public Boo()  {
          Foo f = new Foo();
          f.x = 1;
    }
}
```

Erro aqui porque Foo e Boo estão em pacotes diferentes

Resumindo ...

Visibilidade	Default	Public	Private
A partir da mesma classe	Sim	Sim	Sim
A partir de uma classe do mesmo pacote	Sim	Sim	Não
A partir de uma classe de pacote diferente	Não	Sim	Não

E O PROTECTED ?

Para entender o nível de acesso protegido precisamos, primeiro, falar sobre herança.