Lógica de Programação

Unidade 7 – Encapsulamento e Visibilidade

QI ESCOLAS E FACULDADES
Curso Técnico em Informática

SUMÁRIO

INTRODUÇÃO

No desenvolvimento de *software* orientado a objeto, temos um recurso que auxilia a padronização e controle da criação dos códigos das classes; esse recurso tem o nome de **encapsulamento**. Nesta unidade, vamos aprender sobre este conceito e seus derivados, como a **visibilidade** de atributos e métodos e os métodos **modificadores** e **assessores**.

ENCAPSULAMENTO

De acordo com Scott(2006), mecanismos de encapsulamento permitem que o programador agrupe dados e sub-rotinas(atributos e métodos) em um só lugar, e oculte detalhes sobre a implementação (código) de uma classe.

> *Encapsular significa separar em partes utilizando a abstração (definição do que é realmente relevante). A ideia é deixar o software flexível e facilitar as alterações e o reuso de código (DALE & WEEMS, 2007).*

Imagine que temos uma classe ContaBancaria. Podemos, portanto, possuir no sistema vários objetos desta classe, ou seja, várias contas. Se deixarmos os dados (atributos) públicos, simplesmente **todas** as classes do programa poderão visualizar e modificar qualquer dado de qualquer conta, sem nenhum tipo de controle. Ao ocultarmos os dados, estamos protegendo contra alteração indevida.

É uma prática comum deixar todos os atributos da classe ocultos, para protegê-los. Para entendermos melhor o encapsulamento, vamos analisar o seguinte: **a sua carteira é pública?** Ou seja, qualquer pessoa tem acesso a ela, qualquer um abre e tem acesso ao seu conteúdo?

Provavelmente sua resposta será não, afinal nossa carteira é **privada**, é algo só seu e somente você deve ter acesso a ela, assim você tem o controle do que tem dentro dela. Se alguém quiser algo da sua carteira, terá de pedir a você, certo?

Através do encapsulamento e recursos de visibilidade, o objeto esconde seus dados de outros objetos e permite que os dados sejam acessados por intermédio de seus próprios métodos. Isso é chamado de ocultação de informações (*information hiding*).

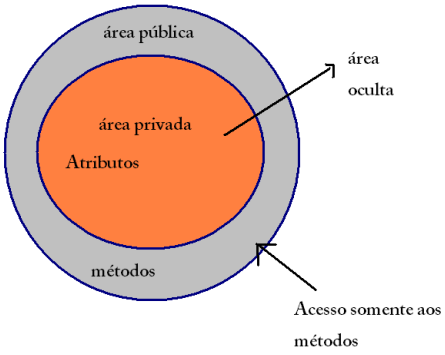


Figura 1[1]

"Cada objeto encapsula uma estrutura de dados e métodos. Uma estrutura de dados encontra-se no centro de um objeto. Os dados do objeto não podem ser acessados, exceto através destes métodos." (MACORATTI)

VISIBILIDADE

Para proteger os dados de uma classe encapsulada, precisamos alterar a sua **visibilidade**.

> *A visibilidade nada mais é do que a maneira que acessamos e enxergamos os dados da nossa classe.*

Quando a visibilidade de um atributo/método é pública ("***public***"), com a qual até o momento estávamos trabalhando, este pode ser visto e acessado de qualquer parte do programa. Já se definirmos a visibilidade como privada ("***private***"), esta só pode ser acessada dentro da própria classe.

> ** Para representarmos a visibilidade de atributos/métodos no diagrama UML, usamos os símbolos + para public e – para private.*

Quando nosso atributo possui a visibilidade pública o acesso a ele é feito de forma direta, ou seja, no Main digitamos o nome do objeto seguido de um ponto, seguido do nome do atributo. Já quando alterarmos a visibilidade para privada, o acesso se torna diferenciado, pois o mesmo não fica visível diretamente pelas outras classes.

[1]Fonte: MACORATTI, José Carlos. http://www.macoratti.net/net_oocb.htm

MÉTODOS ASSESSORES E MODIFICADORES

Todo o atributo que conter visibilidade *private*, terá que possuir dois métodos especiais, um método de acesso **set** para o caso de poder ser alterado, e um método de consulta **get** para o caso de poder ser consultado.

Método set

O *set* é utilizado para que se consiga enviar uma informação para um atributo. Exemplo: informar um nome que será guardado na variável-atributo nome.

O *set* se caracteriza por ser um método **sem retorno**, já que seu objetivo é simplesmente armazenar um dado num atributo, e obrigatoriamente **deve conter argumento**, pois precisa receber um valor externo para poder armazená-lo no atributo.

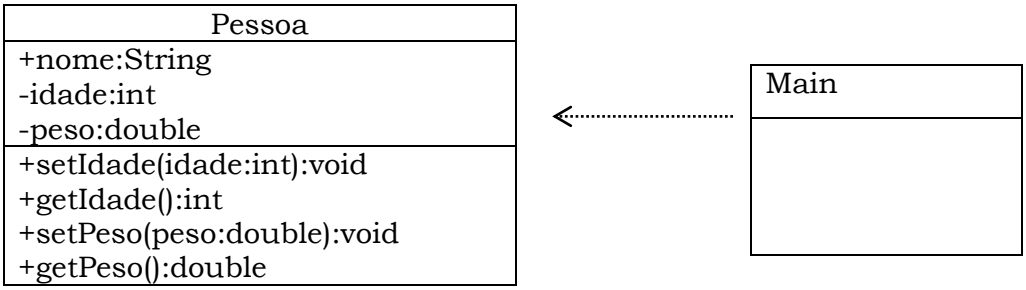
Método get

O *get* é utilizado para consultar/obter o valor de um atributo. Sua função é retornar o valor de um atributo específico. Portanto, **sempre tem retorno**, e **não precisa ter argumentos**.

EXEMPLO DE USO DE ENCAPSULAMENTO

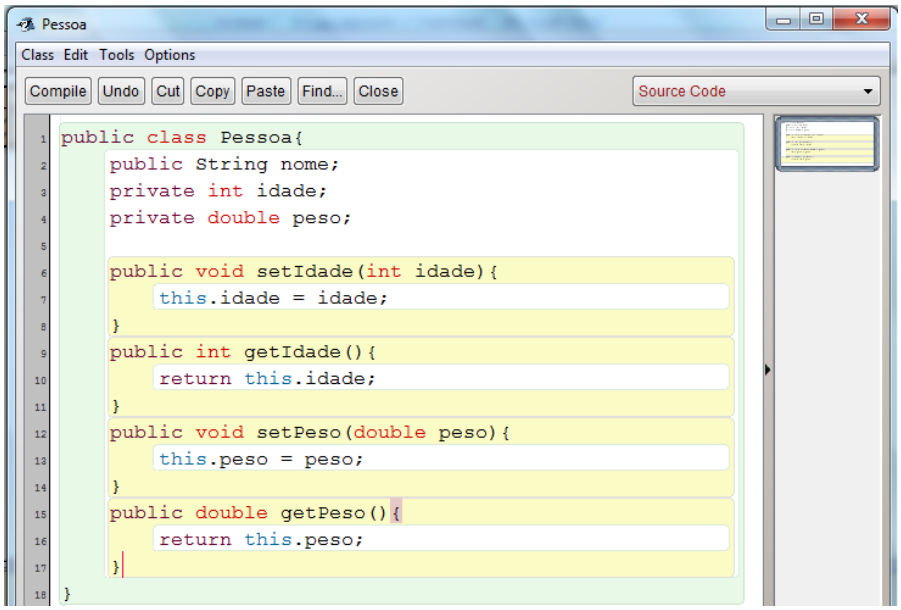
Vamos utilizar como exemplo a classe Pessoa.

Representação de Visibilidade no Diagrama UML



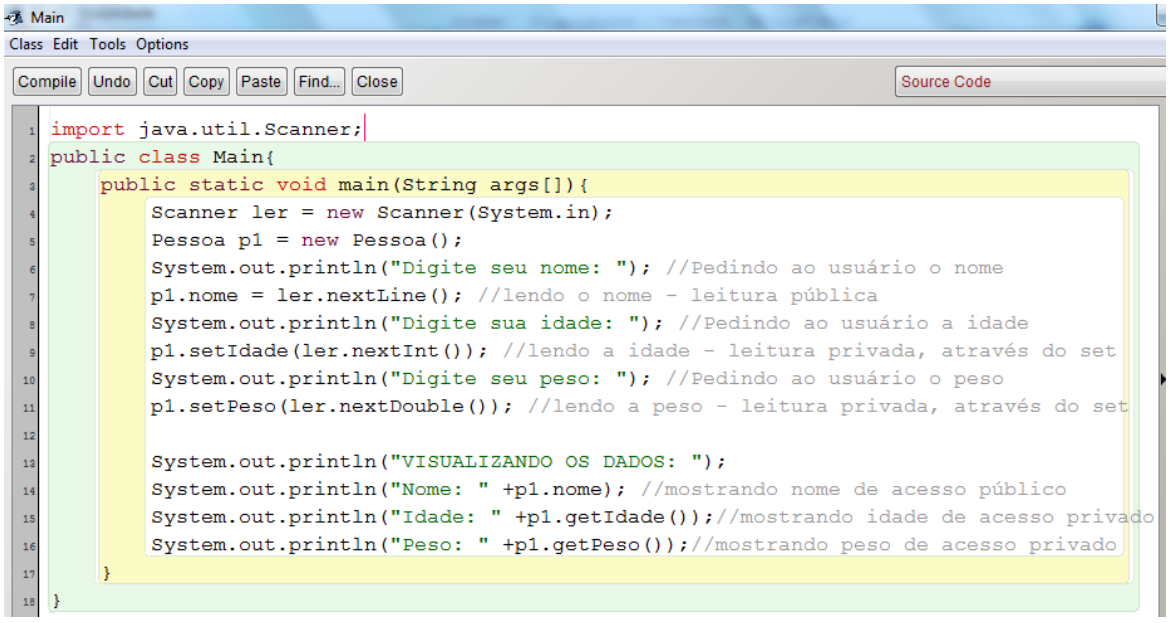
No exemplo acima, percebemos que o atributo **nome** é de acesso público e os atributos **idade** e **peso** são de acessos privados. Para estes últimos, teremos dois métodos especiais, um *setIdade* e *getIdade* e o *setPeso* e *getPeso*. O acesso a estes dados será por intermédio destes métodos, enquanto que o atributo **nome**, por ter sido definido como público, dispensa estes métodos já que o acesso a ele é direto.

Sintaxe da Classe com Encapsulamento



Observando a sintaxe acima, notamos que no método *setIdade* temos um argumento **int idade**, e no método *setPeso*, temos o argumento **double peso**. Através destes argumentos que os dados serão passados aos atributos privados.

Vamos observar a classe Main abaixo, e entender como o acesso é feito.



Observando a classe, notaremos que ao ler o atributo nome, digitamos: p1.nome = ler.nextLine();

```
System.out.println("Digite seu nome: ");
p1.nome = ler.nextLine();
```

Ou seja, inserimos diretamente o nome no atributo da classe, já que foi definido como *public* na classe.

Já em **idade** e **peso**, o acesso foi através do método *set*: p1.setIdade(ler.nextInt());

```
8        System.out.println("Digite sua idade: ");
9        p1.setIdade(ler.nextInt());
```

Inserimos a idade como argumento para o método *setIdade*, este método por sua vez, insere o valor no atributo. O mesmo ocorreu com o atributo peso.

```
10       System.out.println("Digite seu peso: ");
11       p1.setPeso(ler.nextDouble());
```

Ao visualizar os dados, também notamos uma diferença: O atributo nome foi visualizado de forma direta:

```
13       System.out.println("VISUALIZANDO OS DADOS: ");
14       System.out.println("Nome: " +p1.nome);
```

Os atributos idade e peso foram acessados através do *get*:

```
15       System.out.println("Idade: " +p1.getIdade());
16       System.out.println("Peso: " +p1.getPeso());
```

Ou seja, o método **get** foi quem retornou a idade e peso, já que ambos os atributos, por serem privados, não possuem acesso a não ser através dos métodos.

REFERÊNCIAS

MACORATTI, José Carlos. **Conceitos básicos de orientação a objetos**. Disponível em http://www.macoratti.net/net_oocb.htm.

SCOTT, Michael Lee. **Programming language pragmatics**, Edition 2, Morgan Kaufmann, 2006. ISBN 0126339511.

DALE. Nell B; WEEMS, Chip. **Programming and problem solving with Java**, Edition 2, Jones & Bartlett Publishers, 2007. ISBN 0763734020.