TEXTO DE APOIO

AULA 5

Programação de Sistemas I

Professor Regiane Moreno

EaD | Universidade Presbiteriana Mackenzie

Sumário

PROGRAMAÇÃO DE SISTEMAS I

MODIFICADORES DE ACESSO EM JAVA

No paradigma da orientação a objetos, há o conceito de encapsulamento.De acordo com Rafael Santos (2003, p. xx):

> É desejável que os campos das classes sejam ocultos ou escondido de programadores usuários das classes, para evitar que os dados sejam manipulados diretamente em vez de por intermédio dos métodos das classes.

Já vimos que a linguagem Java usa o conceito de encapsular atributos e métodos em um objeto (classe). Ou seja, o encapsulamento “protege” os membros declarados em uma classe e, com isso, permite a restrição de acesso a certas partes de uma classe. Por exemplo, se a classe ContaCorrente tem um atributo saldo, somente os métodos deposito e saque podem alterar o valor do saldo. Não podemos permitir em outra classe que, por meio da instância de um objeto do tipo ContaCorrente, o saldo seja alterado.

Compare os dois casos abaixo:

```
class ContaCorrente {

    String nome;

    double saldo;



ContaCorrente (double saldo){

    this.saldo = saldo;

}

 class TesteBanco{



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

```
    ContaCorrente cta;
cta = new Contacorrente(1000);
 cta.deposito(200);
void deposito (double valor) {
            saldo = saldo + valor;
}

void saque (double valor) {
            saldo = saldo - valor;
}
}
```

Repare que o saldo só é alterado por meiodo construtor e dos métodos da classe ContaCorrente. Porém, da forma como foram declarados os atributos e os métodos, é possível fazer alteração do saldo fora da classe ContaCorrente. Veja:

```
class ContaCorrente {
    String nome;
    double saldo;

ContaCorrente (double saldo){
    this.saldo = saldo;
}
void deposito (double valor) {
```

```
            saldo = saldo + valor;
}

void saque (double valor) {
            saldo = saldo - valor;
}
}
class TesteBanco{

public static void main(String [] args) {
    ContaCorrente cta;
cta = new Contacorrente(1000);

    cta.deposito(200);
cta.saque (100);
cta.saldo = 5000;
}
}
```

Nada impede, da forma como os membros da classe ContaCorrente estão declarados, de se alterar o valor do saldo para R$5000,00 fora da classe.

Os benefícios de se encapsular membros de uma classe são muitos: da clareza do código à minimização de erros e à facilidade de extensão. Talvez a facilidade de modificação seja o mais relevantedos benefícios. Como a classe é conhecida pela sua interface, é muito fácil mudar a representação interna sem que o usuário perceba a diferença. Com encapsulamento, você será capaz de criar componentes de software reutilizáveis, seguros e fáceis de modificar.

Para se usar corretamente o encapsulamento, os membros pertencentes a uma classe devem ser declarados com um dos modificadores de acesso abaixo :

public

Atributos e métodos declarados como ***public*** são visíveis e, portanto, acessíveis a qualquer classe. Esta é a forma menos rígida de encapsulamento, a qual, na verdade, significa não encapsular.

private

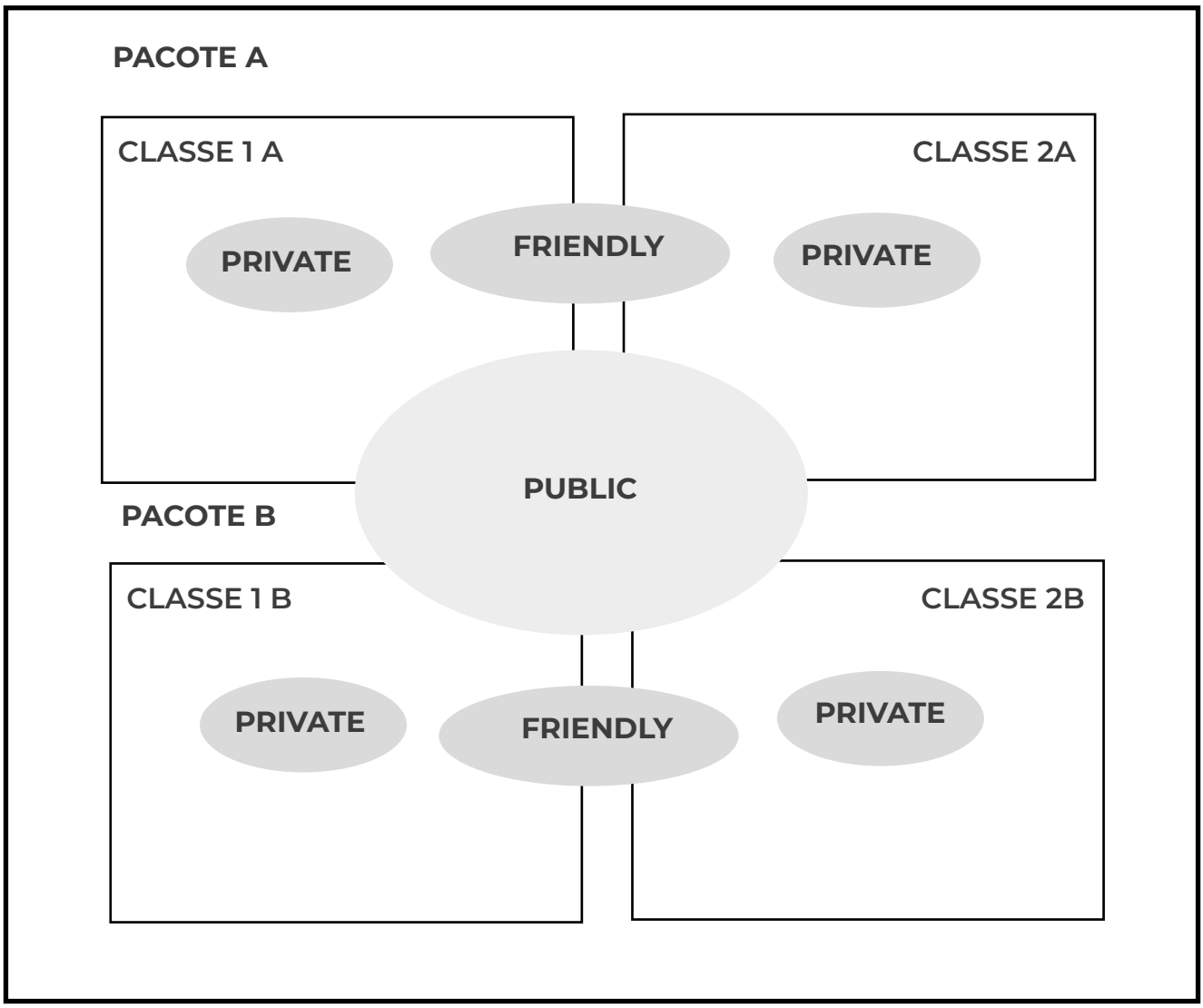
Atributos e métodos declarados como private são visíveis e, portanto, acessíveis apenas na classe onde foram declarados. Ou seja, só podem ser acessados, modificados e/ou executados dentro da própria classe. Esta é a forma mais rígida de encapsulamento.

protected

Atributos e métodos declarados como protected são visíveis e, portanto, acessíveis na classe onde foram declarados e emsuas subclasses. O conceito de subclasse será estudado em Herança.

Sem modificador de acesso

Chamados de “package” ou “friendly”, os membros declarados sem nenhum modificador de acesso são visíveis e, portanto, acessíveis apenas às classes pertencentes ao mesmo pacote.



Fonte: Elaborado pela autora.

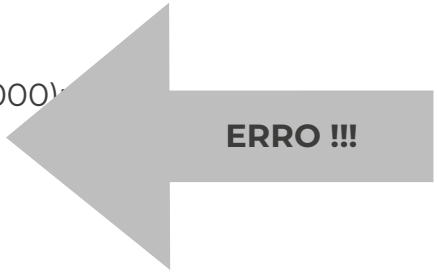
Veja como fica o exemplo anterior, declarando-se os modificadores de acesso:

```
class ContaCorrente {
    String nome;
private double saldo;

ContaCorrente (double saldo){
    this.saldo = saldo;
}
```

```
public void deposito (double valor) {
        saldo = saldo + valor;
}

public void saque (double valor) {
        saldo = saldo - valor;
}
}
```

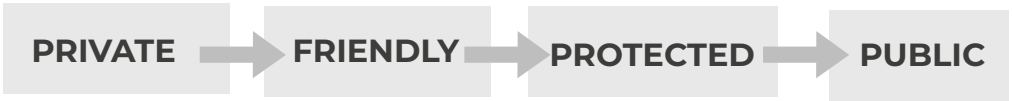
O saldo passou a ser do tipo ***private*** – só é visível e acessível à classe ContaCorrente. Os métodos deposito e saque, por sua vez, são do tipo ***public*** – visíveis e acessíveis a qualquer classe de qualquer pacote. Com esta declaração, seria impossível realizar a alteração do saldo fora da classe ContaCorrente. Ocorreria um erro no trecho seguinte:

```
Class TesteBanco{
 public static void main(String [] args) {
    ContaCorrente cta;
 cta = new Contacorrente(1000);
cta.deposito(200);
```



Observe que, como saldo é private, ele não poderá ser alterado em outra classe.

Assim, temos, do modificador de acesso MAIS restritivo para o MENOS restritivo:



```
public class Teste {
  int a;
  public int b;
  private int c;
void setC (int c){
     this.c = c;
  }
      int getC (){
     return c;
  }
}
public class TestaTeste {
  public static void main(String[] args) {
     Teste obj = new Teste();
     obj.a = 10;
     obj.b = 20;
     // obj.c = 30; não pode ser acessado diretamente
     obj.setC(30);
     System.out.println("a="+ obj.a + " b=" + obj.b+ " c="+ obj.getC());
  }
}
```

GETTERS E SETTERS

Há necessidade de se ter métodos públicos para que se possa acessar os atributos que, em geral, são privados. O padrão adotado pelos programadores em Java para estes métodos é setNomeAtributo() e getNomeAtributo() a fim demodificar e receber os valores dos atributos, respectivamente.

Estes métodos são comumente chamados de Getters e Setters. Um atributo privado pode ter seu conteúdo alterado (set) e recuperado (get) por meiodestes métodos.

Métodos de acesso → getXXX() – permitem o acesso a algum atributo de uma classe.

Métodos modificadores → setXXX() – alteram algum atributo de uma classe.

Apesar de ser possível fazer isso diretamente, de acordo com as boas práticas de programação e o conceito de encapsulamento, essas operações devem sempre ser feitas por meiodestes métodos. Se um atributo é do tipo ***private***, a única forma de alterar e/ou recuperar seu conteúdo é pelos métodos getters e setters.

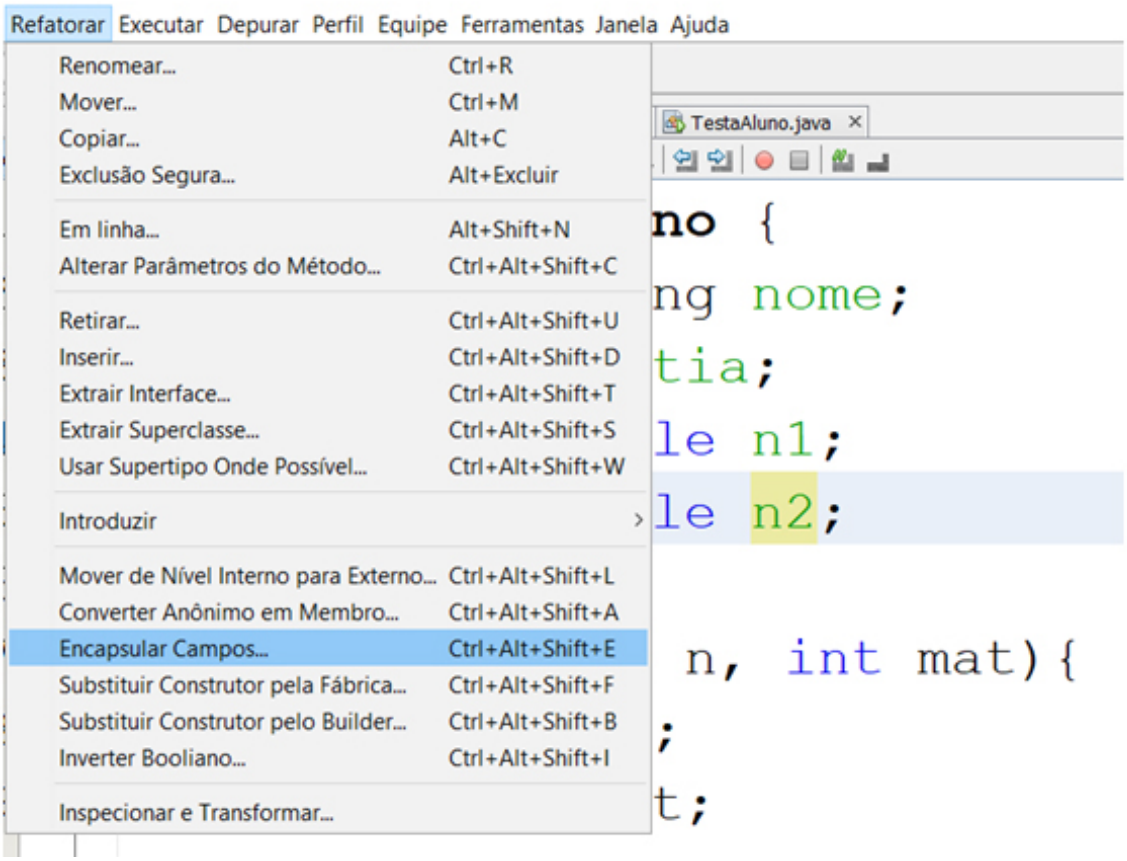
Os programadores (projetistas das classes) não precisam fornecer métodos set() e/ou get() para cada atributo *private*. Essas capacidades devem ser fornecidas somente quando fizerem sentido.

GERANDO AUTOMATICAMENTE GETTERS E SETTERS

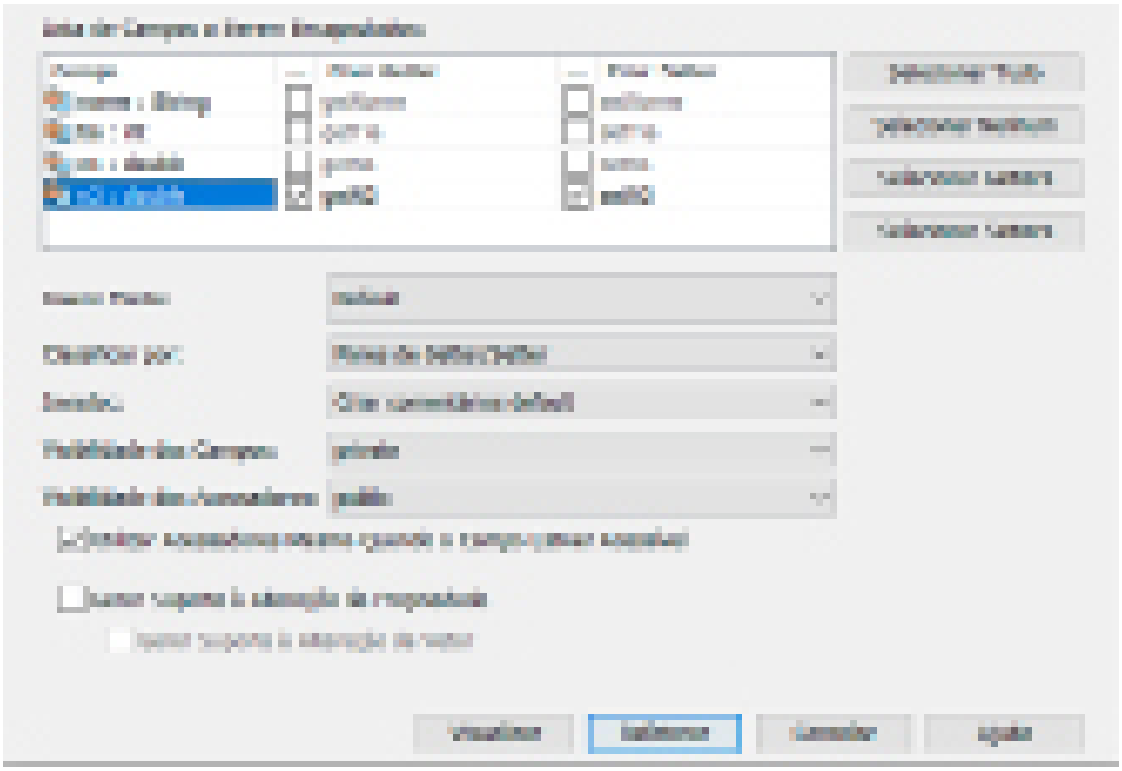
Imagine que você está modelando a classe abaixo e deseja criar os métodos ***Setter/Getter*** para todos os atributos. Por ser uma convenção, existe uma forma de se fazer isso automaticamente, na maioria das IDEs.

Vejamos como fazer isto no NetBeans:

Com a classe aberta, clique no menu *Refatorar* e na opção Encapsular Campos...



Selecione, então, os atributos e os métodos desejados para que a IDE gere o código automaticamente. Veja:



PACOTES EM JAVA

A linguagem Java oferece uma forma de se agrupar as classes em pacotes (em inglês, packages), onde se podem criar grupos de classes que mantêm uma relação entre si. Para a criação destes pacotes, basta se declarar, em cada classe, o pacote a que ela pertence e organizar as classes em diretórios.

A linguagem Java também é organizada por pacotes. Veja um esquema gráfico de uma parte do pacote Java:

Fonte:<https://simplesnippets.tech/packages-in-java/>.

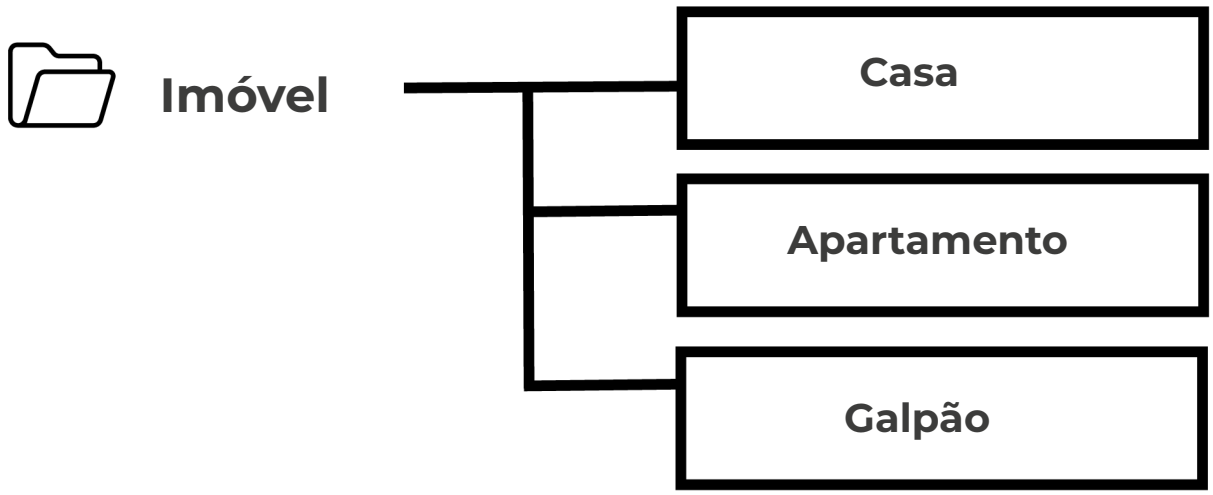
Nesta imagem, podemos observar o pacote java e seus subpacotes lang, util e awt. Veja que o pacote lang contém as classes que equivalem a funcionalidades.

Até o momento, desenvolvemos nossas classes sem mencionar o pacote a que elas pertencem. Quando não se declara um pacote para uma classe, dizemos que o pacote associado é o default. Como todas as classes criadas desta forma pertencem ao mesmo pacote, não são necessárias declarações adicionais quando usamos instâncias de uma classe dentro da outra – basta declarar as instâncias das outras classes que o compilador e máquina virtual se encarregarão de chamar os métodos destas classes.

Complementando, os pacotes são conjuntos de classes relacionadas de forma que elas ofereçam facilidades umas às outras. Estes conjuntos são determinados por meioda codificação de uma linha no topo de cada arquivo, indicando a qual

pacote pertencem as classes ali declaradas. Se nenhuma linha é inserida, assume-se que todas as classes pertencem a um pacote só.

Pacotes requerem que as classes que comporão o pacote sejam armazenadas em um diretório específico. A maneira mais simples de criar um pacote de classes é, então, criar um diretório e colocar lá todos os códigos-fonte das classes que serão consideradas como pertencentes àquele pacote. Vamos supor que criaremos um pacote chamado imovel o qualconterá as classes que representam os diversos tipos de imóveis. Veja como fica a estrutura de diretórios:



O pacote imovel fica armazenado na Pasta (Folder) com o mesmo nome. Os diretórios estão diretamente relacionados aos pacotes e têm, como principal objetivo, reunir as classes que possuema mesma funcionalidade. Assim, se a classe Casa está no pacote imovel, ela deverá estar no diretório imovel. Da mesma forma, se ela se localiza no pacote nacional.saopaulo.imovel, significa que ela está no diretório nacional/saopaulo/imovel. Neste último caso, então, a classe Casa deve iniciar da seguinte forma:

```
package nacional.saopaulo.imovel;

class Casa {

    ...

}
```

Para se ter acesso a uma classe fora do pacote de um determinado programa, deve-se usar o nome longo (exemplo anterior) ou usar a instrução import, que permite uma forma mais fácil de usar as classes definidas em outros pacotes. Não importar um pacote não significa que não se pode usar as classes de outro pacote. Significa que a forma de se chamar as classes deverá ser feita da forma longa, como no exemplo ilustrado acima.

Veja outro exemplo:

```
package br.biblioteca.telas;

import br.biblioteca.Livro;

import br.biblioteca.util.MostrarTela;

import br.biblioteca.util.XmlConvert;

  public class CadastroLivro{

  ..... }
```

Neste caso, a classe CadastroLivro fará parte do pacote telas, que se encontra em br.biblioteca.Esta classe poderá usar a implementação das classes MostraTela e XmLConvert que se encontram no mesmo diretório.

O MÉTODO toString

O método ***toString*** é um dos métodos que toda classe herda direta ou indiretamente da classe *Object* que fica no pacote java.lang.

Este método retorna uma *String*, representando um objeto. Ele é chamado implicitamente sempre que um objeto precisa ser convertido em uma representação *String*.

Você pode utilizá-lo em qualquer classe, todas as classes em Java por default já possuem esse método, mas o programador pode sobrescrever o método da maneira que achar melhor.

Geralmente, o método ***toString*** é utilizado para definir a *String* de saída quando for necessário imprimir o objeto ou quando se desejar convertê-lo para uma determinada *String*.

EXEMPLO:

```
public class Produto {

    int codigo;
    String descricao;
    private double preco;
    private int quantidade;


    public String toString(){
        return "Código:  "+ codigo +"\nDescrição: "+ descricao+ "\nPreco: "+ preco + "\nQuantidade: " + quantidade;
    }
}
```

Observe que o método retorna uma String que apresentará as informações dos atributos da classe.

REFERÊNCIAS

SANTOS, Rafael. Introdução à programação orientada a objetos usando Java. Rio de Janeiro: Elsevier, 2003.