

Encapsulamento e Modificadores de Acesso

Prof. Ms Gustavo Molina

ENCAPSULAMENTO

→ o encapsulamento “protege” os membros declarados em uma classe e com isso permite a restrição de acesso a certas partes de uma classe.

→ **Situação 1:** o atributo `saldo` só é alterado pelos métodos `saque()` e `deposito()`.

```
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;}  
}
```


NO MÉTODO `main()`

```
public static void main(String [] args) {  
    ContaCorrente cta;  
    cta = new Contacorrente(1000);  
  
    cta.deposito(200);  
    cta.saque (100);  
}
```

- o saldo só é alterado através do construtor e dos métodos da classe `ContaCorrente`.
- Porém, é possível alterar o valor do saldo, fora da classe `ContaCorrente`. Veja:

SITUAÇÃO 2

```
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;}  
}  
.  
.  
public static void main(String [] args) {  
    ContaCorrente cta;  
    cta = new Contacorrente(1000);  
  
    cta.deposito(200);  
    cta.saque (100);  
    cta.saldo = 5000;  
}
```



CARACTERÍSTICAS DO ENCAPSULAMENTO

➔ Programar pensando sempre na interface!! Pensar em como os usuários irão utilizar os métodos e não em como os métodos funcionam.

ANALOGIA: Quando você dirige um carro, o que importa são os pedais e o volante (**interface**) e não o motor que você está usando (**implementação**). É claro que um motor diferente pode dar melhores resultados, mas **o que ele faz** é a mesma coisa que um motor menos potente, a diferença está em **como ele faz**. Para trocar um carro a álcool para um a gasolina você não precisa reaprender a dirigir! (trocar a implementação dos métodos não precisa mudar a interface, fazendo com que as outras classes continuem os usando da mesma maneira).

CARACTERÍSTICAS DO ENCAPSULAMENTO

- Programar tendo em vista a **interface** é também chamado de “*programming in the large*”.
- Programar tendo em vista **implementação** (codificação) é chamado de “*programming in the small*”.
- Com encapsulamento você será capaz de criar componentes de software reutilizáveis, seguros, fáceis de modificar.

MODIFICADORES DE ACESSO

public

Atributos e métodos declarados como `public` são visíveis e, portanto, acessíveis a qualquer classe. Esta é a forma menos rígida de encapsulamento, que 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.

MODIFICADORES DE ACESSO

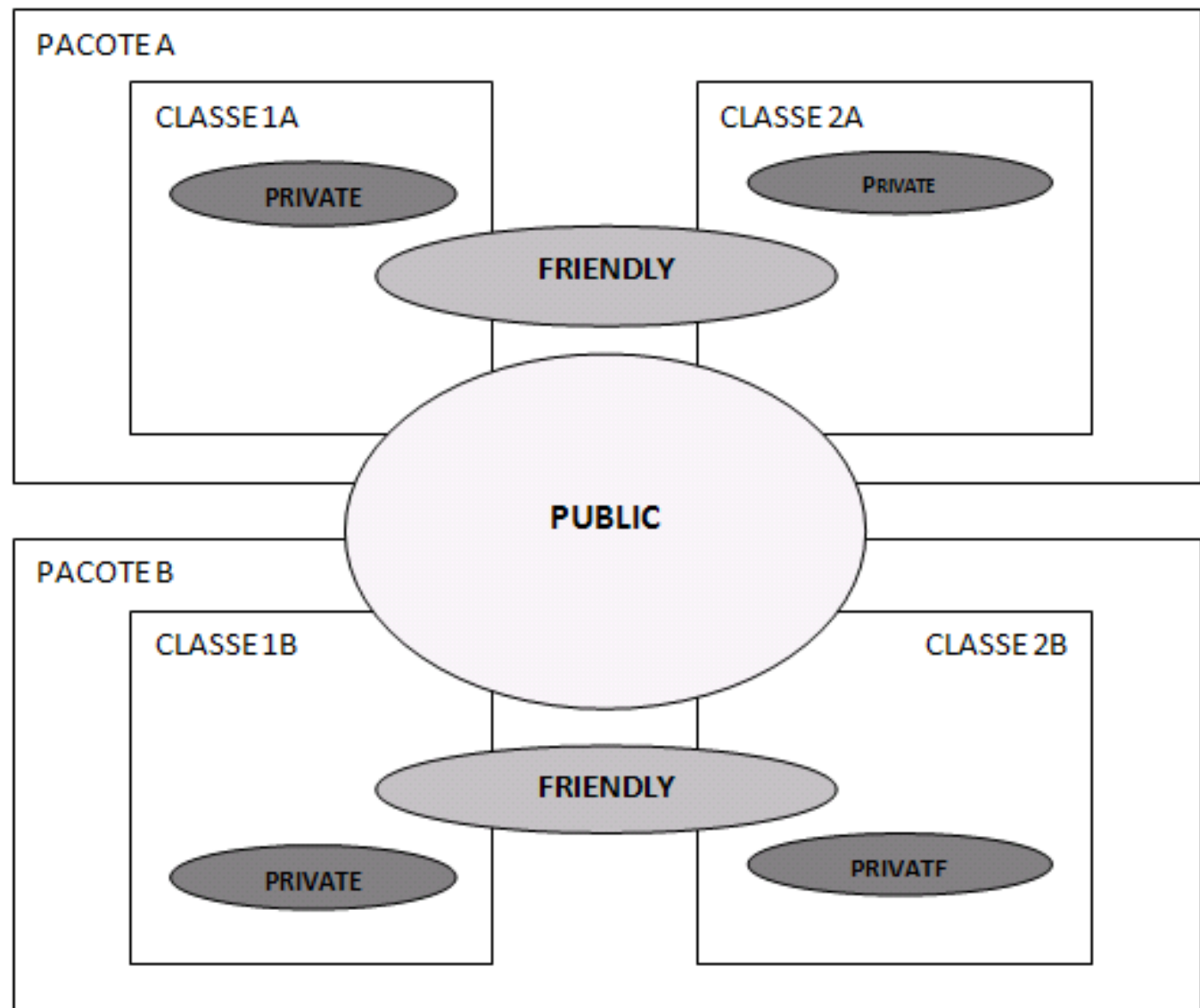
protected

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

Sem modificador de acesso

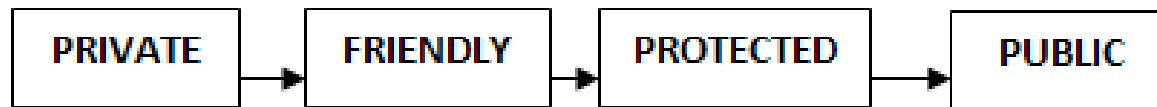
Chamados de “*package*” ou “*friendly*”, os membros declarados sem nenhum modificador de acesso são visíveis e, portanto, acessíveis apenas às classes pertencentes ao mesmo pacote.

Graficamente:



MODIFICADORES DE ACESSO

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



SITUAÇÃO 3 – COM ENCAPSULAMENTO

```
class ContaCorrente {  
    private String nome;  
    private double saldo;  
  
    ContaCorrente (double saldo){  
        this.saldo = saldo;  
    }  
  
    void deposito (double valor) {  
        saldo = saldo + valor;}  
  
    void saque (double valor) {  
        saldo = saldo - valor;}  
}
```

→ O saldo e o nome passaram a ser do tipo `private` – só são visíveis e acessíveis à classe `ContaCorrente`.

→ Os métodos `deposito()` e `saque()` são do tipo `public` – visíveis e acessíveis à qualquer classe de qualquer pacote

SITUAÇÃO 3 – COM ENCAPSULAMENTO

→ seria impossível realizar a alteração do `saldo` fora da classe `ContaCorrente`

```
.  
public static void main(String [] args) {  
    ContaCorrente cta;  
    cta = new Contacorrente(1000);  
  
    cta.deposito(200);  
    cta.saque (100);  
    cta.saldo = 5000;  
}
```



ERRO!!

Setters & Getters

➔ São métodos públicos para acessar atributos privados

```
public class Teste {  
    int a;  
    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());  
    }  
}
```

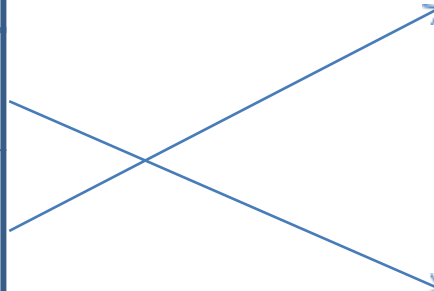
Setter & Getter

<code>int a;</code>
<code>int b;</code>
<code>private int c;</code>
<code>void setC (int c)</code>
<code>int getC ()</code>

É possível acessar / alterar o conteúdo de a e b

Para se ter acesso ao conteúdo de c, deve-se usar o método `getC ()`, que retorna o valor de c

Para alterar o valor de c, deve-se usar o método `setC ()`, recebe um parâmetro e o armazena no atributo protegido



Setters & Getters

Métodos de acesso → `getXXX()` – permitem o acesso à algum atributo de uma classe

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

Padrão adotado, pelos programadores em Java, para estes métodos é `setNomeAtributo()` e `getNomeAtributo()`.