

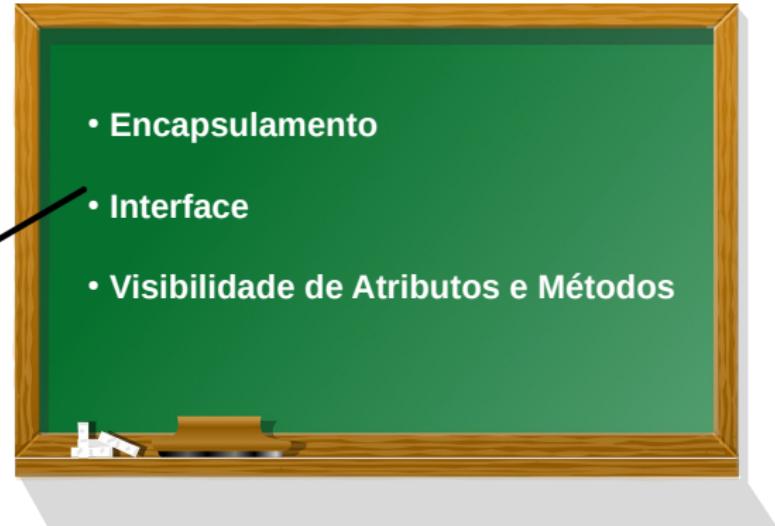
# Encapsulamento e Visibilidade

Luiz Henrique de Campos Merschmann  
Departamento de Computação Aplicada  
Universidade Federal de Lavras

[luiz.hcm@ufla.br](mailto:luiz.hcm@ufla.br)



# Na Aula de Hoje



# Encapsulamento

- ▶ Como vimos anteriormente, os métodos servem:
  - ▶ Para definir o estado de um objeto.
  - ▶ Como mecanismo de comunicação entre objetos.
- ▶ Esconder o estado interno do objeto e requerer que toda interação seja feita por meio dos métodos é chamado de **encapsulamento de dados**.
- ▶ O encapsulamento é um princípio fundamental de OO.



# Encapsulamento

- ▶ Na Orientação a Objetos, as classes (e seus objetos) encapsulam seus dados.
- ▶ Ou seja, elas não permitem que os dados (atributos) sejam diretamente acessados de fora da classe.
- ▶ Nesse caso, a única maneira de acessar ou alterar um dado é por meio dos métodos definidos na própria classe.



- ▶ Resumindo, o objeto está no controle, ou seja, é ele que determina **se** e **como** os atributos podem ser alterados.



# Interface

## Interface de comunicação

- ▶ A interface de comunicação de uma classe é formada pelo seu conjunto de métodos que podem ser chamados de fora da classe.
  - ▶ Esses métodos formam a **interface** de um objeto com o mundo externo, ou seja, por meio deles conseguimos nos comunicar com um objeto.
- ▶ As interfaces permitem informar aos objetos **o que fazer**, mas não **como fazer**.

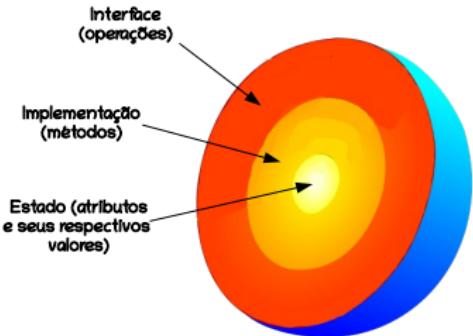
Fazendo uma analogia com uma máquina de cartão:

- ▶ Os botões da máquina são a interface entre nós e os mecanismos internos do aparelho.
- ▶ Ou seja, podemos usar uma máquina de cartão pressionando seus botões sem saber como é o seu funcionamento interno.



# Interface e Encapsulamento

- ▶ O encapsulamento garante que a única maneira de **acessar** ou **alterar** dados de um objeto seja através da sua **interface de comunicação**.
- ▶ O encapsulamento garante o **ocultamento de informações**.



- ▶ O princípio da ocultação estabelece que devemos expor apenas a interface de comunicação entre os objetos.

# Interface

Fazer com que a única forma de se comunicar com um objeto e acessar seus dados seja por meio dos métodos traz duas vantagens:

- ▶ Mantendo a interface de comunicação (**assinaturas dos métodos**), você pode **alterar** a representação interna dos dados **sem afetar** o restante do sistema.
  - ▶ Exemplo: Inicialmente você usava vetor como um tipo de dado. Depois, por problemas de gerenciamento de memória, você troca o vetor por uma lista encadeada. Se a interface for mantida, isso não causará qualquer efeito colateral.
- ▶ Para alguém **conseguir usar um objeto** de uma classe definida por você, basta **fornecer a interface de comunicação**, ou seja, as assinaturas dos métodos.

# Encapsulamento em Java

Uma vez que Java é uma linguagem de programação orientada a objetos, então o encapsulamento de dados está garantido? Ou seja, os atributos de uma classe em Java são inacessíveis fora dela?

- ▶ A princípio não.

```
1 class ContaBancaria
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta
```

```
luiz@HP:~/TesteProgramas$ javac ContaBancariaTeste.java
luiz@HP:~/TesteProgramas$ java ContaBancariaTeste
Novo saldo da conta: R$ 100,00
```

```
1 /**
2  * Classe principal que nos permite utilizar
3  * objetos da classe ContaBancaria
4  */
5 public class ContaBancariaTeste
6 {
7     public static void main(String[] args)
8     {
9         ContaBancaria minhaConta = new ContaBancaria(1234, "Luiz Henrique");
10        //Acessando o valor de um atributo fora da classe ContaBancaria!!!
11        minhaConta.saldo = 100;
12        System.out.printf("Novo saldo da conta: R$ %.2f\n", minhaConta.getSaldo());
13    }
14 }
```

- ▶ Mas uma classe pode definir o tipo de acesso que seus membros permitirão às demais partes do sistema.

# Encapsulamento em Java

Para encapsular um atributo em uma classe Java, precisamos **explicitar** isso no código.

- ▶ Os modificadores de acesso *public* e *private* controlam o acesso aos atributos da classe.

```
class ContaBancaria
{
    //atributos
    int numero; //numero da conta
    String nome; //nome do titular da conta
    float saldo; //saldo da conta
```

```
class ContaBancaria
{
    //atributos
    private int numero; //numero da conta
    private String nome; //nome do titular da conta
    private float saldo; //saldo da conta
```



- ▶ A palavra-chave *private* é utilizada para indicar que os atributos são **privados** da classe. O que isso significa?
  - ▶ Significa que a única forma de saber o valor dos atributos ou alterá-los, de fora da classe, é através dos métodos da classe.
- ▶ **Não utilizar** o modificador de acesso *private* para os atributos de uma classe **fere o princípio do encapsulamento**.

# Visibilidade de Atributos e Métodos

- ▶ Além do modificador de acesso denominado *private*, existe também o *public*. Qual será a finalidade desse segundo modificador?
  - ▶ Como é possível de imaginar, o *public* define que um membro é **acessível fora** da sua classe.
  - ▶ E membro significa **atributo** ou **método**.
- ▶ Portanto, os modificadores de acesso definem a **visibilidade** de um **atributo ou método** de uma classe.

```
1 class ContaBancaria
2 {
3     //atributos
4     private int numero; //numero da conta
5     private String nome; //nome do titular
6     private float saldo; //saldo da conta;
7
8     //Retorna o numero da conta
9     public int getNumero()
10    {
11        return numero;
12    }
13    //...demais métodos
14 }
```

Desse modo, se desejamos que um método possa ser chamado de fora da sua classe, temos que inserir no código o modificador *public*.

# Métodos Privados

Mas faz algum sentido usar um método privado?

- ▶ Sim, métodos privados são muito interessantes por dois motivos:
  - ▶ Em algumas situações percebemos que mais de um método público de nossa classe necessita de um mesmo trecho de código. A fim de não repetir esse trecho de código em mais de um método público da classe, separa-se esse trecho em um método privado que passa a ser chamado nos métodos públicos.
  - ▶ Um método público poderia ficar muito grande, o que dificultaria a legibilidade e manutenção do código. Nesse caso, pode-se dividir esse método em partes menores fazendo-se uso de métodos privados.

# Métodos *Get* e *Set*

- ▶ Já vimos que o modificador *private* faz com que a única forma de saber o valor de um atributo ou alterá-lo, de fora da classe, seja através dos métodos dessa classe.
- ▶ Portanto, uma prática comum para acessar os atributos privados é utilizar:
  - ▶ Métodos *set*: são métodos modificadores, pois geralmente mudam o estado de um objeto.
  - ▶ Métodos *get*: são métodos de acesso, que permitem consultar os valores dos atributos.
- ▶ Comumente o nome desses métodos é dado pelo prefixo (**get** ou **set**) seguido pelo nome do atributo em questão (utilizando-se a notação *camelCase*).

# Métodos *Get* e *Set* x Atributos Públicos

Fornecer os métodos *get* e *set* é o mesmo que tornar *public* os atributos da classe?

- ▶ Pode parecer que sim, mas há uma diferença sutil.
- ▶ Se o atributo for declarado *private*, o método *get* (*public*) permitirá o acesso ao atributo a partir de outras classes, mas ele poderá controlar como esse dado será acessado.
  - ▶ Por exemplo, o método *get* pode controlar o formato dos dados que ele retorna.
- ▶ De modo semelhante, o método *set* (*public*) pode, e deve, examinar cuidadosamente as tentativas de modificar o valor de um atributo.
  - ▶ Por exemplo, tentativas de definir a idade de uma pessoa com valor negativo devem ser rejeitadas.
- ▶ Resumindo, apesar de os métodos *get* e *set* fornecerem acesso a atributos *private*, esse acesso é **controlado** pela implementação dos métodos!

# Métodos *Get* e *Set*

Devo criar métodos *get* e *set* para todos atributos (*private*) da classe?

- ▶ Analise o nosso exemplo da aula anterior, onde temos a classe **ContaBancaria**:
  - ▶ Os atributos dessa classe são: o nome do titular, o número da conta e o saldo da mesma.
  - ▶ Além disso, ela possui métodos que permitem o cliente movimentar a sua conta (*sacar* e *depositar*).
  - ▶ Faz sentido um método *set* para alterar o atributo saldo?
  - ▶ Na vida real, como o saldo de uma conta é alterado?
- ▶ Portanto, devemos criar métodos *set* e *get* somente quando for realmente necessário!

# Uma Falha Comum em OO

Suponha que uma classe *Cena* é formada por um chão, um sol e uma pessoa, sendo estes objetos das classes *Quadrado*, *Circulo* e *Pessoa*, respectivamente. Existe também um método *get* para retornar o sol da cena.

- ▶ O que acontece se fora da classe eu declarar uma variável que recebe o retorno do método *getSol* e chamar o método *mudarCor* usando essa variável? Isso vai alterar a cor do sol da cena em si?
  - ▶ Sim, vai alterar a cor do sol da cena. Isso acontece porque o método retorna uma referência (endereço) do sol da cena.

# Uma Falha Comum em OO

Mas qual é o problema disso?

- ▶ Nós aprendemos com o conceito de encapsulamento que a classe deve ser a única capaz de alterar os seus dados.
- ▶ Dessa forma, se uma cena possui um círculo (*sol*), os dados desse sol não deveriam ser alterados fora da classe *Cena*.

Como evitar isso?

- ▶ Nesses casos deve ser retornada uma cópia do objeto (*sol*) e não a referência para o objeto em si.
- ▶ A linguagem Java padroniza um nome de método (*clone*) para retornar uma cópia para o objeto.
- ▶ Mas se você precisasse implementar um método desse sem conhecer o *clone*, como você faria com o que já conhece da linguagem?

# Perguntas?

