Uniube

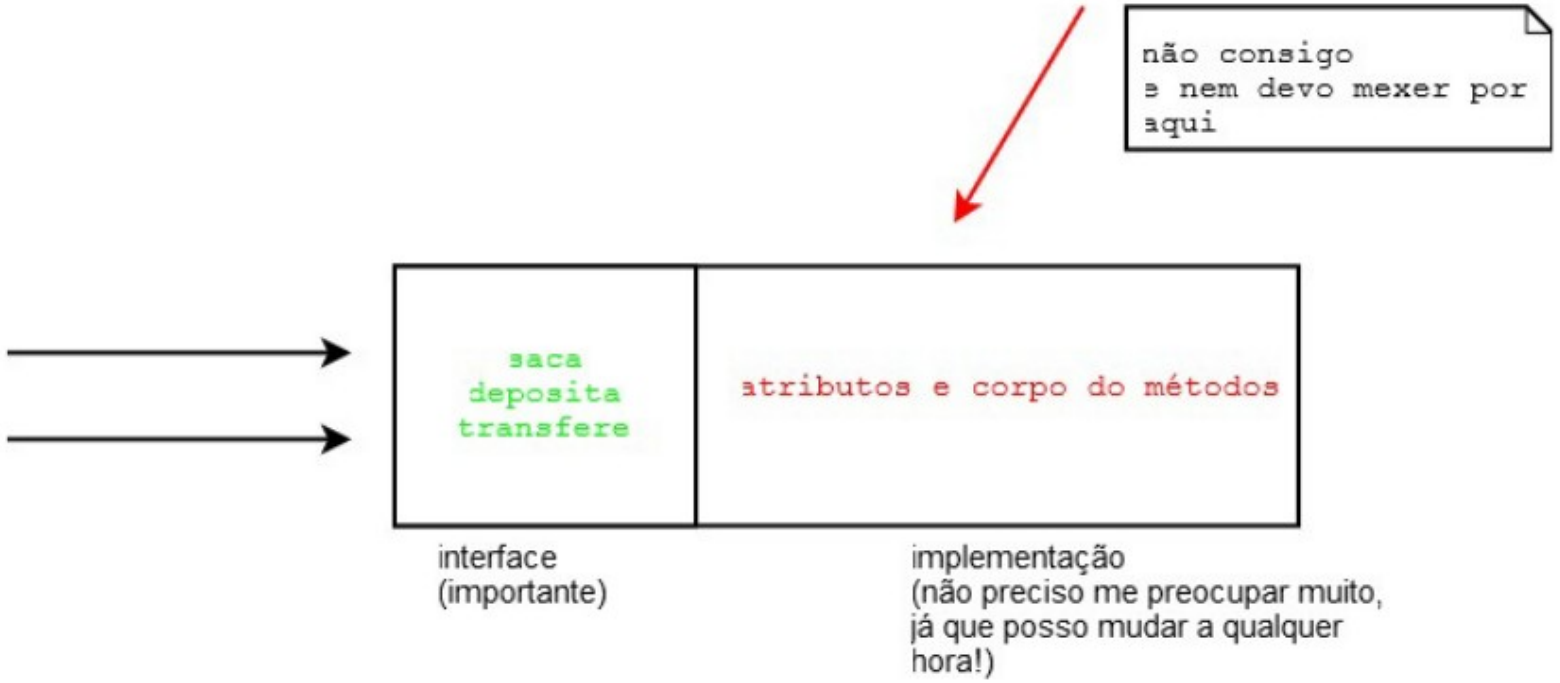
Programação Orientada a Objetos

Prof. Me. Clênio Silva
clenio.silva@uniube.br

Encapsulamento

- Encapsular, isto é ocultar todos os membros de uma classe, além de esconder como funcionam as rotinas (no caso, métodos) do sistema.
- Encapsular é fundamental para seu sistema ser suscetível a mudanças:
  - Não precisaremos mudar a regra de negócio em vários lugares, mas sim, em apenas um único lugar, já que essa regra esta encapsulada.

Encapsulamento



O conjunto de métodos públicos de uma classe é também chamado de **interface da classe**, pois essa é a única maneira pela qual você se comunica com objetos dessa classe.

Encapsulamento

O modificador **private** faz com que ninguém consiga modificar e tampouco ler o atributo em questão. Com isso, temos um problema: como fazer para mostrar o saldo de uma Conta, uma vez que nem mesmo podemos acessá ló para leitura?

Encapsulamento

```
class Conta {

    private double saldo;

    // outros atributos omitidos

    public double pegaSaldo() {
        return this.saldo;
    }

    // deposita() e saca() omitidos
}
```

```
class TestaAcessoComPegaSaldo {
   public static void main(String[] args) {
    Conta minhaConta = new Conta();
    minhaConta.deposita(1000);
    System.out.println("Saldo: " + minhaConta.pegaSaldo());
   }
}
```

Encapsulamento – Getters e Setters

- A fim de permitir o acesso aos atributos (Já que eles são **private**) de uma maneira controlada, a prática mais comum é criar dois métodos, um que retorna o valor, e outro o qual muda o valor.
- A convenção para esses métodos é de colocar a palavra **get** ou **set** antes do nome do atributo. Por exemplo, a nossa conta com `saldo`, `limite` e `titular` fica assim caso desejamos dar o acesso da leitura e escrita a todos atributos:

Encapsulamento – Getters e Setters

```
class Conta {

    private String titular;
    private double saldo;

    public double getSaldo() {
        return this.saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public String getTitular() {
        return this.titular;
    }

    public void setTitular(String titular) {
        this.titular = titular;
    }
}
```

Construtores

- Quando usamos a palavra-chave **new**, estamos construindo um objeto. Sempre quando o **new** é chamado, ele executa o construtor da classe. O construtor da classe é um bloco declarado com o mesmo nome que a classe:

```
class Conta {
    String titular;
    int numero;
    double saldo;

    // construtor
    Conta() {
        System.out.println("Construindo uma conta.");
    }

    // ..
}
```

Então, quando fizermos:

```
Conta c = new Conta();
```

A mensagem "construindo uma conta" aparecerá. É como uma rotina de inicialização que é chamada sempre que um novo objeto é criado. Um construtor pode parecer, mas **não é** um método.

Construtores

Até agoras, as nossas classes não tinham nenhum construtor. Então, como é que era possível dar **new** se todo **new** chama um construtor **obrigatoriamente**?

Quando você não declara nenhum construtor na sua classe, o Java cria um para você. Esse construtor é o **construtor default**. Ele não recebe nenhum argumento e o seu corpo é vazio.

Construtores

- Um construtor pode receber argumentos, inicializando, assim, algum tipo de informação:

```
class Conta {
    String titular;
    int numero;
    double saldo;

    // construtor
    Conta(String titular) {
        this.titular = titular;
    }

    // ..
}
```

Esse construtor recebe o titular da conta. Dessa maneira, quando criarmos uma conta, ele já terá um determinado titular.

```
String carlos = "Carlos";
Conta c = new Conta(carlos);
System.out.println(c.titular);
```

Construtores

Tudo estava funcionando até agora. Para que utilizamos um construtor?
A ideia é bem simples. Se toda conta precisa de um titular, como obrigar todos os objetos que forem criados a ter um valor desse tipo? É só criar um único construtor que receba essa String!

O construtor se resume a isso! Dar possibilidades ou obrigar o usuário de uma classe a passar argumentos para o objeto durante o seu processo de criação.

Praticando…