

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Linguagem de Programação I

Introdução à OO

- ▶ Tiago Maritan
- ▶ tiago@ci.ufpb.br

Motivação

- ▶ Até agora aprendemos a programar apenas usando o **paradigma de programação estruturada (imperativa)**
- ▶ Nesse paradigma:
 - ▶ Programa é um **conjunto de métodos (funções)**;
 - ▶ Dados (variáveis) são usados para apoiar as funções;
 - ▶ Difícil construir programas grandes (complexos);
- ▶ Hoje começaremos a aprender um novo paradigma: o **paradigma de programação orientada a objetos (POO)**;

Motivação

- ▶ Para onde quer que olhemos no mundo real, vemos (e pensamos) em termos de **objetos!!!**
 - ▶ Pessoas, animais, plantas, carros, aviões, computador, livros, edifícios, etc.



- ▶ Ex: **Sala de aula:** alunos, cadeiras, mesas, lousa, etc.

Motivação

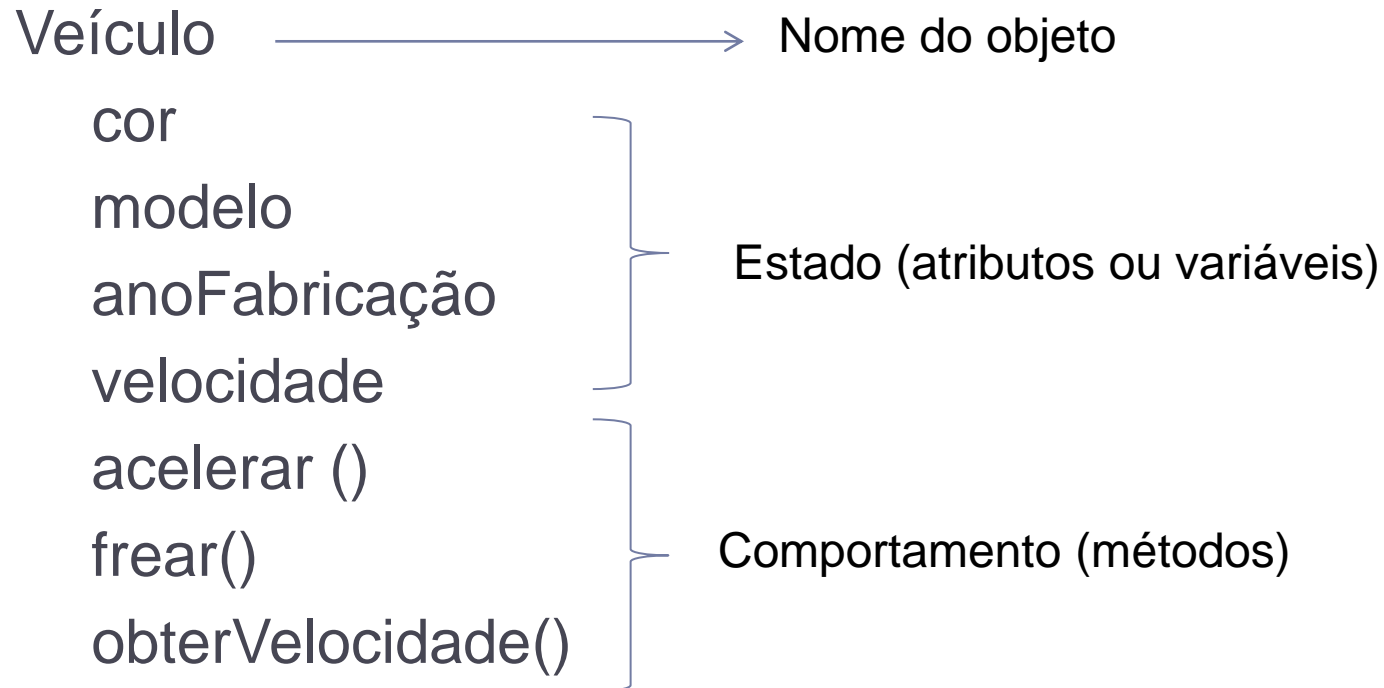
- ▶ Esses objetos possuem em comum:
 - ▶ **Estados (ou atributos)**: Ex: tamanho, forma, cor e peso
 - ▶ **Comportamentos (ou operações)** : Ex: a bola rola, incha, esvazia; o bebê chora, dorme, engatinha; o professor leciona, etc.
- ▶ Desde pequenos, aprendemos sobre objetos estudando seus atributos e observando seus comportamentos
 - ▶ Ex: Comparar bebês e adultos, humanos e macacos;
 - ▶ Ex: Carros, caminhões e patins de rodas têm muito em comum;

Objetos

- ▶ A OO se inspira nessas idéias para modelar os programas como ***um conjunto de “objetos” que se relacionam!***
 - ▶ **Objetos** são abstrações de **objetos reais** existentes;
 - ▶ Ex: alunos, cadeiras, lousa, mesa, etc.
 - ▶ Objetos compartilham 2 características:
 - ▶ **Estado (atributos)**: conjunto de atributos definido por meio de variáveis.
 - ▶ Ex: cor, modelo, ano de fabricação de um veículo, etc...
 - ▶ **Comportamento (operações)**: conjunto de métodos que ele possui.
-
- ▶ 5 ▶ Ex: acelerar(), frear()...

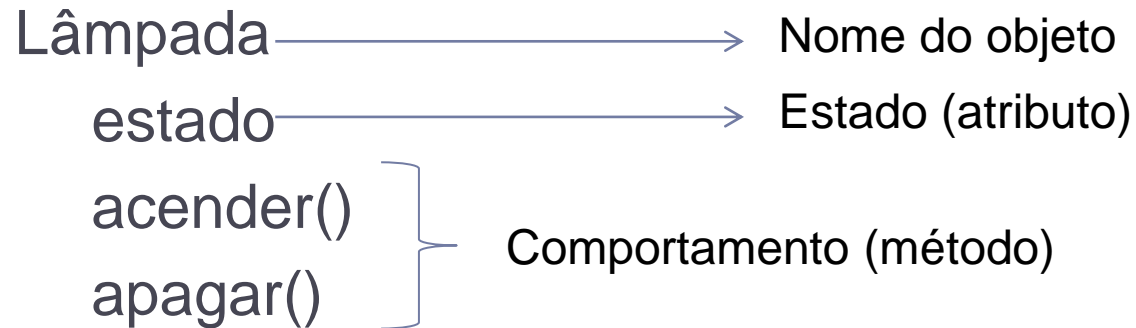
Objetos

► Exemplo 1: Objetos Veículo



Objetos

► Exemplo 2: Objetos Lâmpada

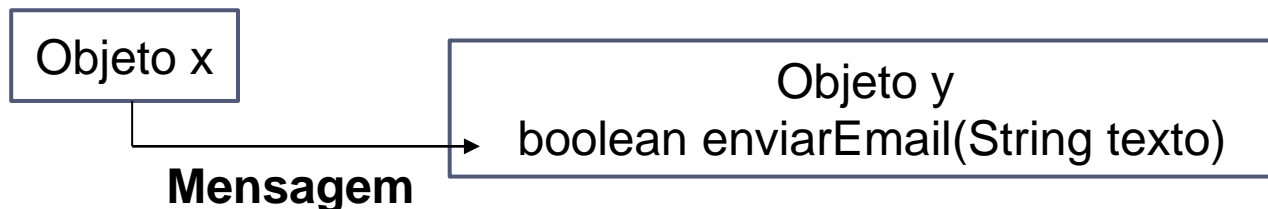


Objetos

- ▶ Exercício: Definir **Objetos Quadrado**.

Comunicação entre Objetos

- ▶ Um **sistema** pode ser composto por **muitos objetos**;
- ▶ Esses objetos se comunicam através de **mensagens**;
 - ▶ Um objeto solicita a outro que este execute algum método.
- ▶ **Ex:** **Objeto X** solicita ao **objeto Y** que ele envie um e-mail.



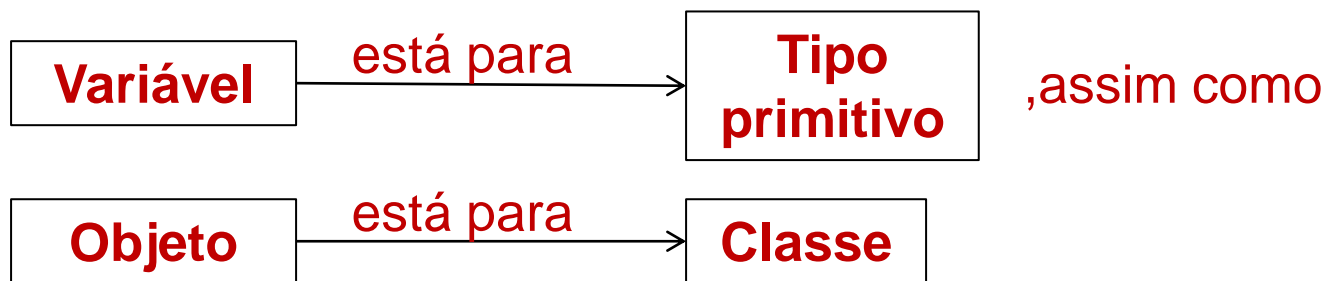
Diz-se que **x enviou uma mensagem a y**.
(a mensagem pode possuir **parâmetros** e **retorno**)

Classes

- ▶ **Objetos** são criados a partir das **classes**.
 - ▶ Ou seja, objetos são **instâncias** em memória de uma classe;
- ▶ **Classe é um molde, um modelo, um protótipo a partir do qual os objetos podem ser criados**
 - ▶ Ex: Imagine a **classe** como um **projeto de um veículo** (partes que o compõem, listas de peças, ações que podem ser executadas);
 - ▶ Depois de criado o projeto... **muitos objetos veículos** podem ser criados com base nesse projeto (**classe**).
- ▶ Ou seja, pode se criar **vários objetos** a partir de uma **classe**;

Classes

- ▶ Uma **classe** também pode ser vista como um **tipo definido pelo programador**



- ▶ Exemplo:

```
// x eh uma variavel do tipo int
int x = 10;

// y eh um objeto da classe Automovel
Veiculo y = new Veiculo();
```

Criação de uma classe

► Sintaxe:

```
<qualificador> class <Nome-da-classe>{  
    // atributos (estados)  
    // métodos (operações)  
}
```

- <qualificador>: indica o nível de acesso da classe. Ex:
 - **private**: classe só pode ser usada no mesmo pacote (diretório);
 - **public**: classe pode ser usado livremente por qualquer classe;
 - Se não for declarado nada, a classe só é visível em seu próprio pacote

Criação de uma Classe

► Exemplo: Classe Veículo

```
public class Veiculo {  
    String cor = "Preta";  
    String modelo = "Palio";  
    int anoFabricacao = 2011;  
    double velocidade = 0.0;  
  
    public void acelerar(double aumVel){  
        velocidade += aumVel;  
    }  
  
    public void frear(double dimVel){  
        velocidade -= dimVel;  
    }  
}
```

Utilização de Objetos

► Como vimos, uma **classe** permite criar **vários objetos**

► Para se utilizar um objeto é necessário:

1. **Declarar o objeto:** Semelhante a declaração de uma variável;

Sintaxe: `<Nome-classe> <nome-objeto>;`

Exemplo: `Veiculo veiculo1;`

2. **Instanciar e inicializar o objeto:** Alocar memória para o objeto e definir valores iniciais para as variáveis do objeto.

► Para inicializar é utilizado o método **construtor (veremos adiante)**;

Sintaxe: `<nome-objeto> = new <construtor>;`

Exemplo: `veiculo1 = new Veiculo();`

Exemplo 1

► Criação da classe Produto

```
public class Produto {  
    String nome;  
    double preco;  
  
    // declaração de métodos  
}
```

Exemplo 1

► Uso de Objetos da classe Produto

```
public class UsaProduto {  
  
    public static void main(String args[]) {  
        Produto produto1 = new Produto();  
        Produto produto2 = new Produto();  
  
        produto1.nome = "Shampoo";  
        produto1.preco = 1.34;  
  
        produto2.nome = "Sabonete";  
        produto2.preco = 2.58;  
    }  
}
```


Exemplo2

► Criação da classe Veiculo

```
public class Veiculo {  
    String cor = "Preta";  
    String modelo = "Palio";  
    int anoFabricacao = 2011;  
    double velocidade = 0.0;  
  
    public void acelerar(double aumVel) {  
        velocidade += aumVel;  
    }  
  
    public void frear(double dimVel) {  
        velocidade -= dimVel;  
    }  
    public double obterVelocidade() {  
        return velocidade;  
    }  
}
```

Exemplo2

► Uso de Objetos da classe Veiculo

```
public class UsaVeiculo{

    public static void main(String args[]){
        Veiculo veiculo1 = new Veiculo();
        Veiculo veiculo2 = new Veiculo();

        veiculo1.cor = "Vermelha";
        veiculo2.ano = 2007;

        veiculo1.acelerar(100.0);
        veiculo2.acelerar(50.0);

        veiculo1.frear(20.0);
        double v1 = veiculo1.obterVelocidade();
        System.out.println(v1);
    }
}
```

Convenção de nomes para objetos e classes

- ▶ **Nome da classe**, em geral, inicia-se com **letra maiúscula**
- ▶ **Variáveis e objetos** iniciam seu nome com **letra minúscula**
- ▶ **Métodos e atributos** iniciam seu nome com letra minúscula
- ▶ É comum nomear as variáveis objetos com o mesmo nome da classe e um índice.

- ▶ Ex: `Veiculo veiculo1, veiculo2, veiculo3;`

- ▶ Ex: `Pessoa pessoa1, pessoa2, pessoa3;`

Construtores

Construtores

- ▶ São **métodos especiais** invocados **no momento da criação dos objetos** (inicialização);

```
Veiculo v1 = new Veiculo(); // invocação do  
//construtor padrão Veiculo()
```

- ▶ Existe um **construtor padrão**: não precisa ser programado;
- ▶ É o **primeiro método** que um objeto executa:
 - ▶ Garante a **inicialização correta do objeto**.
 - ▶ Ex: Na criação de janelas (GUI), construtor pode definir: cor de fundo, tamanho dos botões, etc.



Construtores

- ▶ **Construtor padrão:**

- ▶ Inicializa dados do objeto com **valores default** para tipos primitivos ou `null` (para objetos)

Tipo primitivo	Valor Padrão
boolean	false
char	espaço
Tipos numéricos	Zero do tipo (0, 00)

- ▶ Quando o programador define um construtor, o construtor padrão não pode ser invocado;



Construtores

- ▶ Construtores x outros métodos:
 - ▶ Os construtores têm o **mesmo nome da classe**.
 - ▶ Os construtores **não retornam valor** (nem mesmo `void`). Devem ser declarados sem retorno.
 - ▶ Não podem ser chamados sem o **new**.
 - ▶ Exceto se for chamado por outro construtor, com o `this`;



Construtores

► Definindo vários construtores

```
public class Cliente{  
    int codigo;  
    String nome, cidade;  
  
    public Cliente() {} // 1o construtor  
  
    public Cliente(int cod) { // 2o construtor  
        codigo = cod;  
    }  
  
    public Cliente(int cod, String n) { // 3o construtor  
        codigo = cod;  
        nome = n;  
        cidade = "Indaiatuba";  
    }  
}
```


Construtores

► Usando os diferentes construtores;

```
public class UsaCliente{

    public static void main(String args[]){
        Cliente c1 = new Cliente();
        Cliente c2 = new Cliente(1);
        Cliente c3 = new Cliente(2, "Tatiana");

        System.out.println(c1.codigo+ "-" + c1.nome);
        System.out.println(c2.codigo+ "-" + c2.nome);
        System.out.println(c3.codigo+ "-" + c3.nome);
        System.out.println(c4.codigo+ "-" + c4.nome);

    }
}
```

Uso da palavra reservada `this`

- ▶ A palavra reservada **this** faz referência ao objeto corrente, isto é, ao objeto usado no momento.
- ▶ Exemplo:

```
float velocidade;  
public void setVelocidade(float velocidade) {  
    this.velocidade = velocidade;  
}
```

- ▶ O **this** nesse caso permite diferenciar as duas variáveis: a variável de instância declarada antes e o parâmetro passado

Uso da palavra reservada `this`

```
public class Cliente{
    int codigo;
    String nome, cidade;

    public Cliente(int codigo){
        this.codigo = codigo;
    }

    public Cliente(int codigo, String nome){
        this.codigo = codigo;
        this.nome = nome;
        this.cidade = "Indaiatuba";
    }
}
```

Uso da palavra reservada `this`

- ▶ A invocação de Construtores

- ▶ Apenas construtores podem chamar construtores

- ▶ Outros métodos não podem;

- ▶ Construtores são chamados usando **`this`**;

- ▶ Se um construtor for chamado a partir de outro, esta deve ser a primeira linha de código do chamador;



Uso da palavra reservada `this`

```
public class Cliente{
    int codigo;
    String nome, cidade;

    public Cliente(int codigo){
        this.codigo = codigo;
    }

    public Cliente(int codigo, String nome){
        this(codigo); // chamada ao 1o construtor
        this.nome = nome;
        this.cidade = "Indaiatuba";
    }
}
```

Destrutores

- ▶ Maioria das linguagens OO também definem **destrutores**;
- ▶ Responsáveis por liberar os recursos usados pelos objetos;
 - ▶ “Passa a borracha” nos endereços de memória alocados para o objeto
- ▶ Construtores -> alocam espaço, destrutores -> desalocam
- ▶ Destrutores são desnecessários em Java, pois Java possui um **coletor automático de lixo** (*automatic garbage collector*)

Coletor Automático de Lixo

- ▶ Processo automático para limpeza de objetos que não estão mais sendo utilizados.

▶ Ex:

```
Cliente c1;  
c1 = new Cliente("Tatiana");  
c1 = new Cliente("Izadora");
```

← Espaço perdido na memória.
Será marcado para eliminação

- ▶ Coletor de lixo é invocado quando o processo está ocioso;

Encapsulamento



Encapsulamento (*Data Hiding*)

- ▶ Importante conceito da OO;
- ▶ Permite **restringir o acesso** a **variáveis** e **métodos** da classe
- ▶ **Ocultar** certos **detalhes de implementação** do usuário da classe.
 - ▶ Usuário usa os serviços sem saber como ocorre internamente;
 - ▶ Exemplo: Liquidificador: dona de casa não precisa saber como o liquidificador funciona internamente...
 - ▶ Ele só precisa conhecer as interfaces: ligar() e desligar();

Encapsulamento (*Data hiding*)

- ▶ Ele também **minimiza os erros de programação**;
 - ▶ Evitando, por exemplo, **quebra de integridade dos dados**;
 - ▶ Ex: pode evitar que usuários alterem diretamente uma variável saldo de um objeto ContaBancária.
- ▶ Facilita a **atualização do código**.
 - ▶ Fabricante da classe pode modificar internamente a classe sem que o usuário da classe se dê conta disso;
 - ▶ Ex: fabricante do liquidificador poderia mudar o motor para um mais moderno...
 - ▶ ... usuário da classe continuaria precisando apenas da interface para manipular o novo liquidificador: `ligar()` e `desligar()`;

Encapsulamento (*Data Hiding*)

- ▶ Para determinar o **nível de acesso** dos elementos da classe (variáveis e métodos) utiliza-se os qualificadores:
 - ▶ **public**: nível sem restrições, equivalente a não encapsular;
 - ▶ **private**: nível mais rígido, apenas a própria classe pode acessar.
 - ▶ **protected**: nível intermediário, apenas a própria classe e as subclasses podem acessar. Classes do mesmo pacote também podem.
 - ▶ **package**: apenas classes do mesmo pacote podem acessar.


Encapsulamento (*Data Hiding*)

```
public class Veiculo {  
    public String nome = "";  
    private float velocidade = 0.0;  
  
    public void acelerar(double aumVel){  
        velocidade += aumVel;  
    }  
  
    public void frear(double dimVel){  
        velocidade -= dimVel;  
    }  
}
```

Encapsulamento (*Data Hiding*)

Nível de acesso **private**.
Variável encapsulada.

```
public class Veiculo {  
    public String nome = "";  
    private float velocidade = 0.0;  
  
    public void acelerar(double aumVel){  
        velocidade += aumVel;  
    }  
  
    public void frear(double dimVel){  
        velocidade -= dimVel;  
    }  
}
```



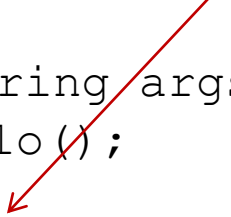
Encapsulamento (*Data Hiding*)

```
public class UsaVeiculo2 {  
    public static void main(String args[]){  
        Veiculo v1 = new Veiculo();  
        v1.nome = "Gol"; // ok  
        v1.velocidade = 80.0; // erro de compilação  
  
        v1.acelerar(50.0); // ok  
    }  
}
```

Encapsulamento (*Data Hiding*)

Classe UsaVeiculo não pode acessar
atributo velocidade

```
public class UsaVeiculo2 {  
    public static void main(String args[]){  
        Veiculo v1 = new Veiculo();  
        v1.nome = "Gol"; // ok  
        v1.velocidade = 80.0; // erro de compilação  
  
        v1.acelerar(50.0); // ok  
    }  
}
```



Métodos `get()` e `set()`

- ▶ Os atributos de uma classe são, em geral, `private`;
 - ▶ Outras classes não podem acessar esses atributos;
 - ▶ Boa prática de programação;
- ▶ As classes costumam fornecer métodos `public` para permitir que outras classes **configurar** ou **obter** atributos `private`;
 - ▶ Métodos `get()`: Obtém (recuperam) o valor de uma variável
 - ▶ Ex: `getCor()`, `getModelo()`
 - ▶ Métodos `set()`: Configuram (definem) o valor de uma variável.
 - ▶ Ex: `setCor(String cor)`, `setModelo(String modelo)`

Métodos get () e set ()

```
public class Produto {  
    private String nome;  
    private double preco;  
  
    public String getNome() {  
        return nome;  
    }  
    public double getPreco() {  
        return preco;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public void setPreco(double preco) {  
        this.preco = preco;  
    }  
}
```

Métodos get () e set ()

```
public class UsaProduto {  
    public static void main(String args) {  
        Produto p1 = new Produto();  
        p1.setNome("Shampoo");  
        p1.setPreco(2.34);  
  
        System.out.println("Nome =" + p1.getNome());  
        System.out.println("Preco = " + p1.getPreco());  
  
    }  
}
```

Exercício

- ▶ Reescreva a classe Veiculo criando os construtores, métodos get() e set(), considerando que a classe possui os seguintes atributos:
 - ▶ cor, modelo, ano e velocidade

Campos e Métodos estáticos

Entendendo Membros de Instância e de Classe

- ▶ Com “**static**” criamos atributos e métodos que pertencem à **classe** e não a uma **instância da classe (objeto)**
- ▶ Quando vários objetos são criados a partir da mesma classe (fôrma), cada um possui sua própria cópia distinta das **variáveis de instância**
 - ▶ Ex: nome, email, telefone de Pessoa
- ▶ Quando se quer algumas variáveis que sejam **comuns a todos os objetos**, usa-se o modificador “static”
 - ▶ Ex: Mesma `taxaJuros` para todas as `AgenciaBancaria`



Atributos estáticos

- ▶ Campos compartilhados por todas os objetos
 - ▶ São declarados com a palavra reservada `static`
- ▶ Exemplo: Taxa de juros a ser utilizada por todas as agencias bancárias

```
public class Agencia {  
    private String endereco;  
    private int nome;  
    public static double juros = 0.4;  
  
    public String getEndereco() {...}  
    public void ligarAlarme() {...}  
}
```

Métodos Estáticos

- ▶ Métodos que podem ser executados sem que objetos sejam criados

```
NomeDaClasse.nomeDoMetodo(argumentos)
```

- ▶ Ex: `Integer.parseInt("30");`
`JOptionPane.showInputDialog("message");`
`main(String args[])`
- ▶ Também é declarado usando `static`.
- ▶ Não conseguem acessar **variáveis de instância** e **métodos de instância** diretamente
 - ▶ Ex: `main()` só chama diretamente métodos estáticos

Pacotes

Pacotes

- ▶ Java permite que classes sejam agrupadas em **pacotes**.
- ▶ Um pacote (package) em Java é um diretório (pasta) em que está armazenada uma ou mais classes
 - ▶ Isto é, um **pacote** é um **conjunto de classes**;
- ▶ Normalmente as classes de um pacote tem um mesmo propósito.
 - ▶ Ex1: `java.net` – conjuntos de classes para trabalhar em rede;
 - ▶ Ex2: `javax.swing` – conjunto de classes para GUI;
- ▶ Todas as classes pertencem a algum pacote. Quando este não é especificado, diz-se que a classe pertence ao pacote *default*

Pacotes

- ▶ Declara-se o pacote de uma classe com a declaração na primeira linha de uma classe:

```
package nomedopacote;
```

- ▶ Exemplo:

```
package meupacote;  
  
public class Veiculo{  
    // código da classe  
}
```

**Classe Veiculo e Bicicleta
pertencem a meupacote**

```
package meupacote;  
  
public class Bicicleta {  
    // código da classe  
}
```

-
- ▶ Para usar uma classe de um outro pacote, devemos usar a diretiva **import**:

```
import nomedopacote.NomeDaClasse;
```

- ▶ Ou então para importar todas as classes de uma vez só:

```
import nomedopacote.*;
```

- ▶ Ex:

```
import meupacote.Veiculo;

public class Teste{
    Veiculo v1; //
}
```

Modelagem OO: Usando de Diagramas de Classes

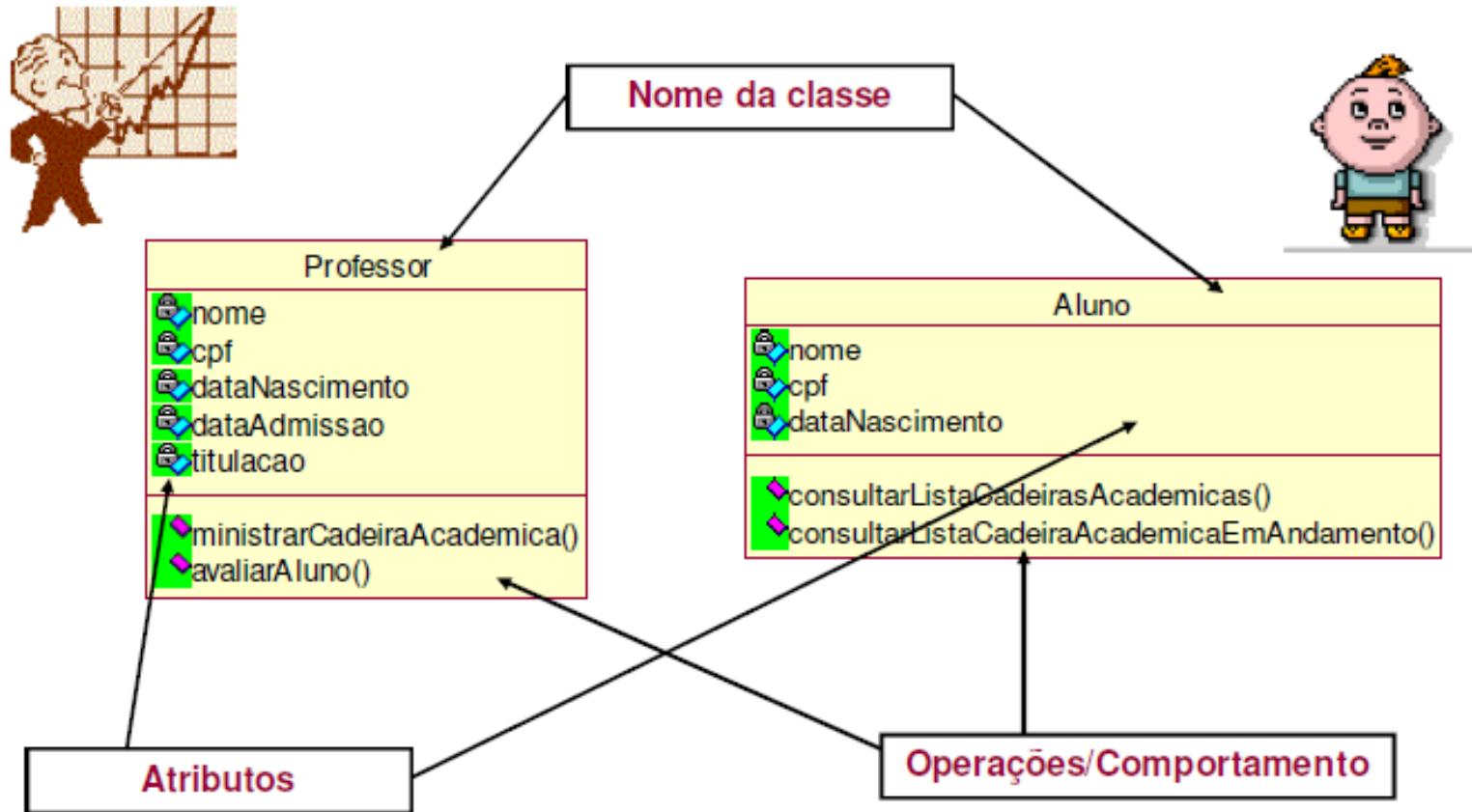
Diagrama de Classes

- ▶ **Descreve:**
 - ▶ tipos de objetos no sistema
 - ▶ e os vários tipos de relacionamento estático entre eles
- ▶ **Mostram propriedades de uma classe**
 - ▶ Atributos,
 - ▶ Métodos
 - ▶ Associações com outras classes.

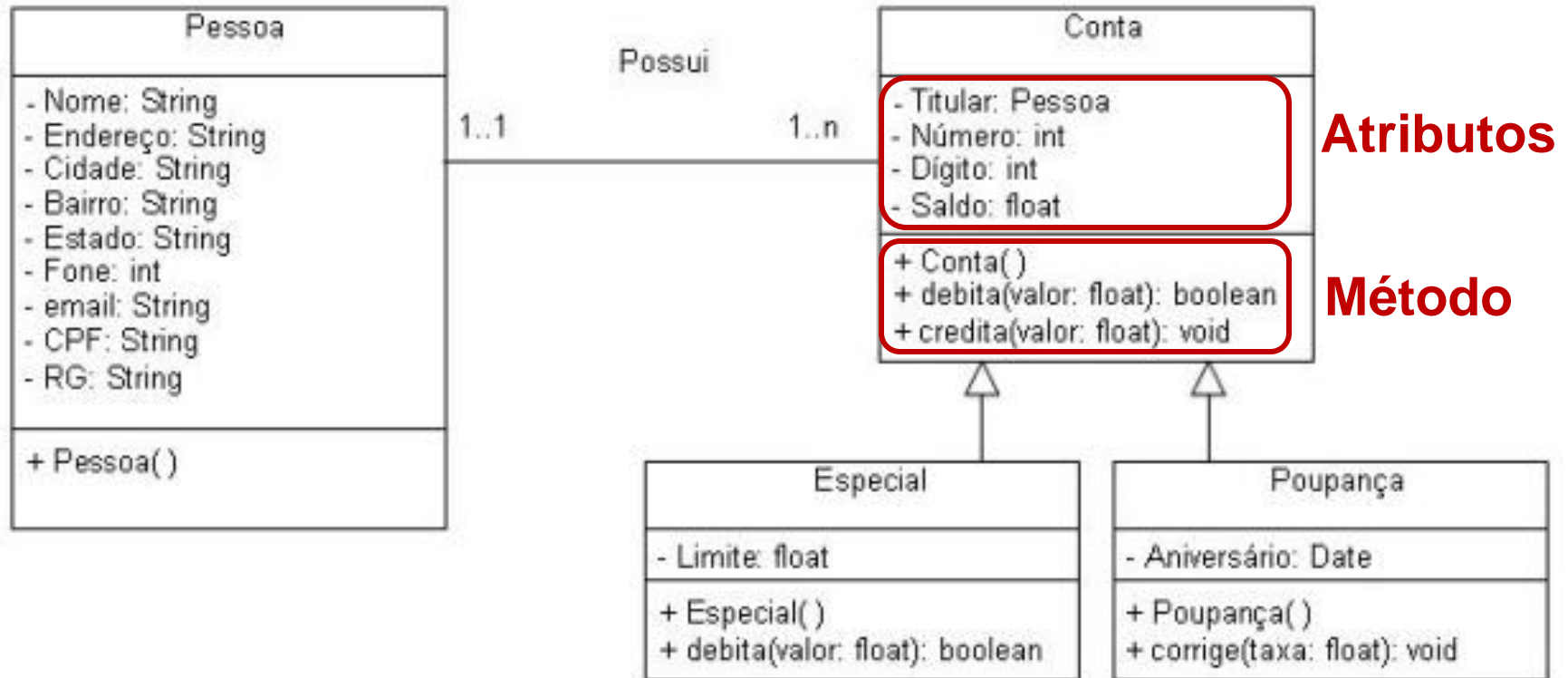
Caixas do Diagrama de Classes

- ▶ As caixas representam classes
- ▶ Cada classe é dividida em 3 compartimentos:
 - ▶ Nome da classe
 - ▶ Atributos
 - ▶ Operações da classe
- ▶ As setas representam a relação entre as classes
 - ▶ Associação, composição, herança, etc.

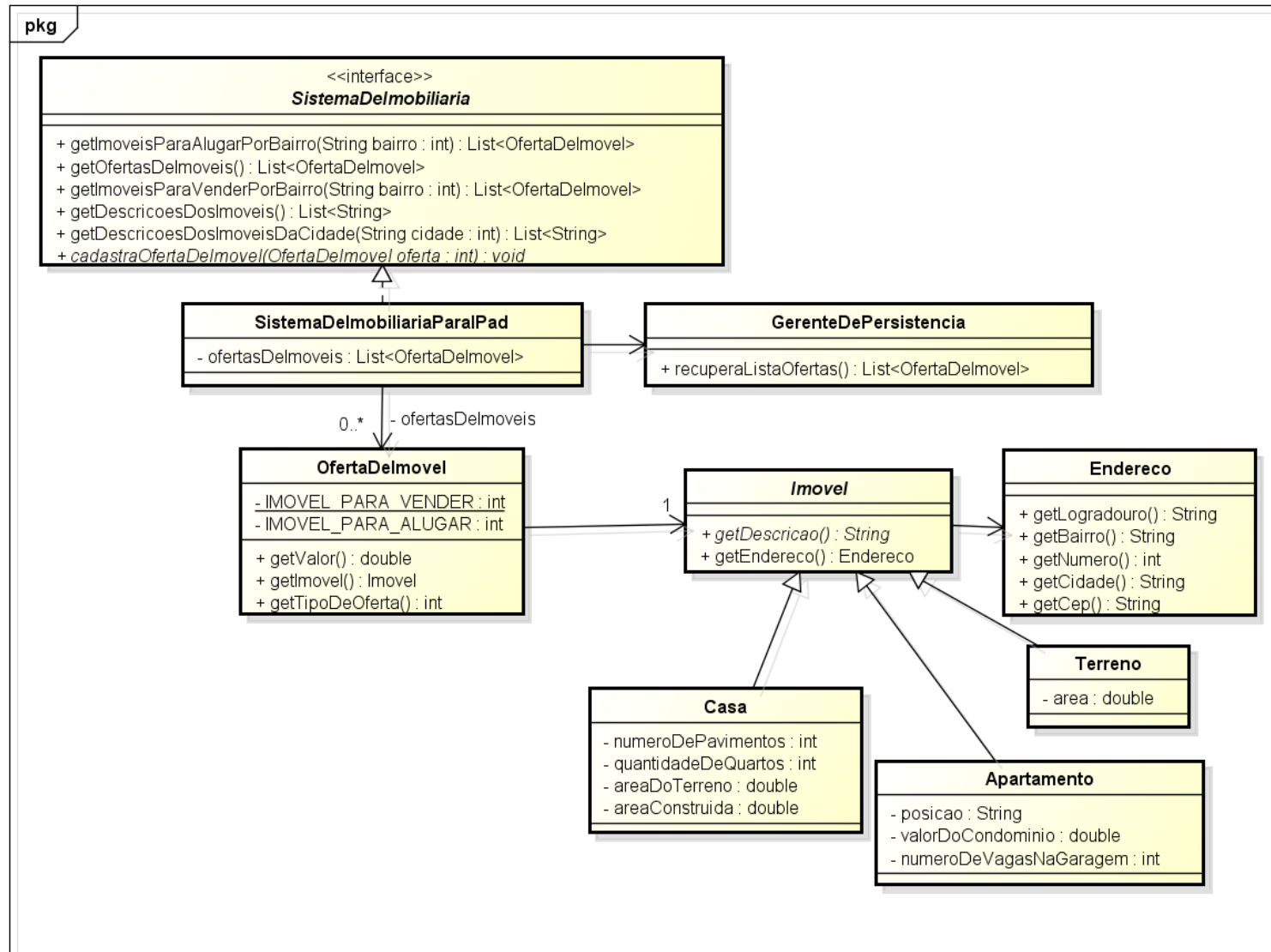
Exemplo



Ex: Aplicação Bancária



Ex: Sistema de Imobiliária



Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Linguagem de Programação I

Introdução à OO

- ▶ Tiago Maritan
- ▶ tiago@ci.ufpb.br