

A collection of historical artifacts is arranged on a light-colored surface. In the top left, a portion of a wooden chessboard with a checkered pattern and several chess pieces is visible. Below it, a blue ribbon with a circular medal is displayed. To the right, a red ribbon with a circular medal is shown. Further right, a white star-shaped medal with a central emblem is visible. In the bottom left, a round compass with a white face and black markings is shown. A pair of thin-framed glasses lies horizontally across the lower middle of the image.

Introdução aos Conceitos OO

Profª. Roberta B Torres (rbtorresiff@gmail.com)

Material: sites.google.com/site/rbtorresiff



Conceitos Gerais

□ POO – Programação Orientada a Objetos

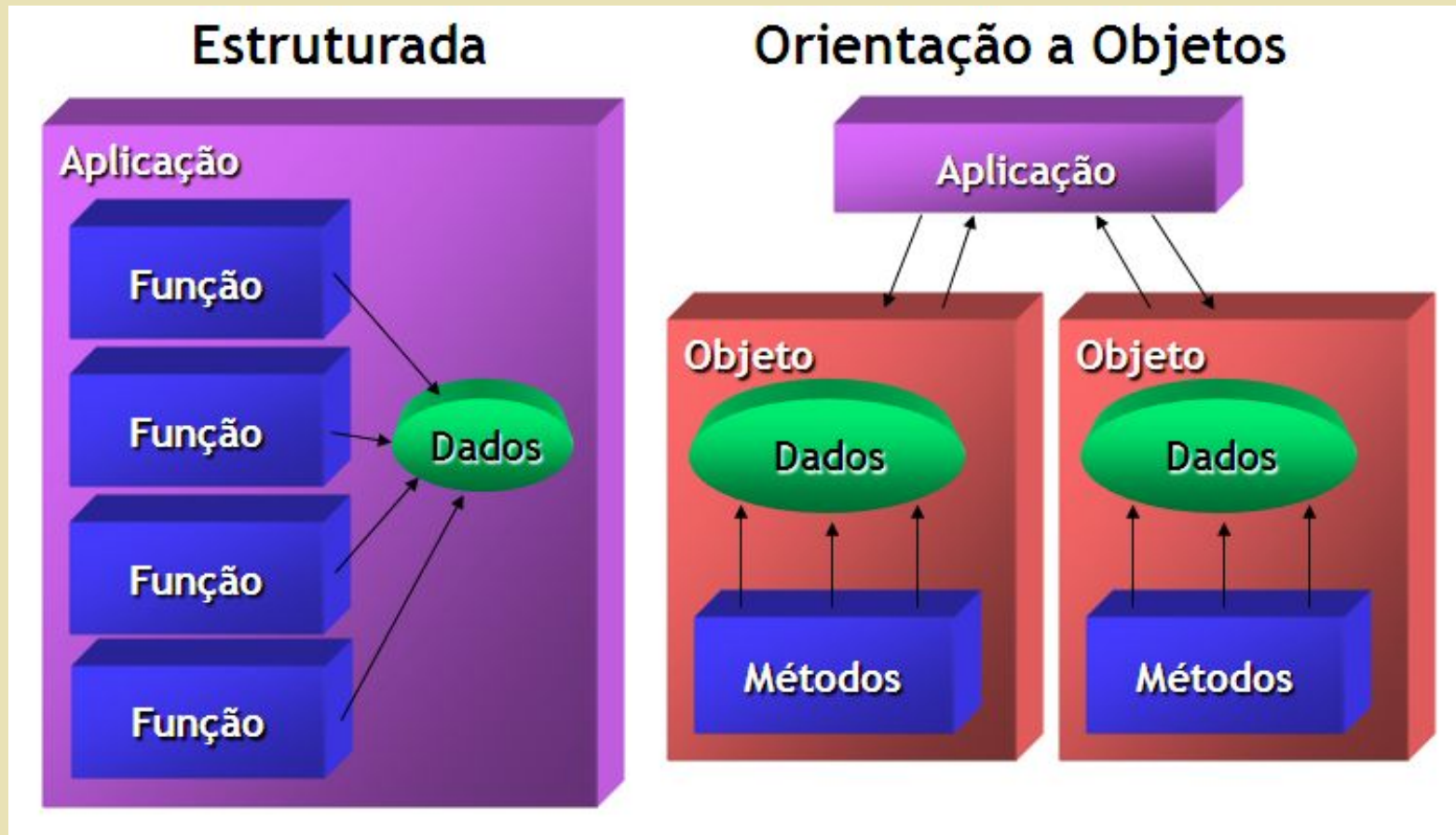
- É um paradigma na forma de pensar, modelar e implementar software; mais um estilo de programação.
- Uma forma de desenvolver software através da simulação de objetos do mundo real.
- Possui um conjunto de métodos, técnicas e ferramentas que permite desenvolver componentes de software reusáveis e mais facilmente manuteníveis.

Evolução das técnicas de programação

1950 – 1960 Era do Caos	1970 – 1980 Era da Estruturação	1990 até agora Era dos Objetos
Saltos, gotos, variáveis não estruturadas, variáveis espalhadas ao longo do programa	If-then-else Blocos Registros Laços-While	Objetos Mensagens Métodos Herança

Conceitos Gerais

Introdução à POO





Conceitos Gerais

□ Introdução à POO

DIFICULDADES

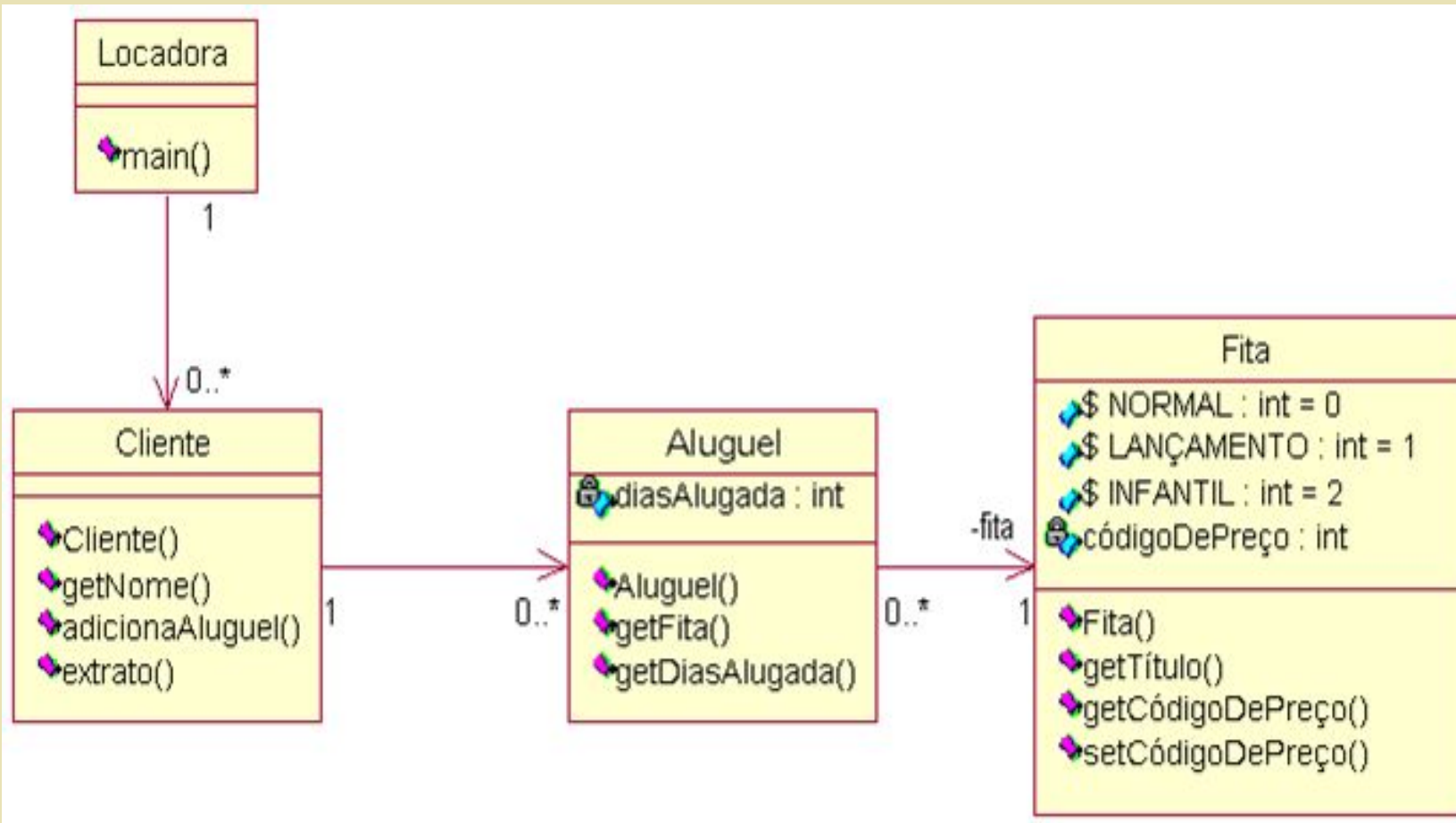
- Complexidade no aprendizado em comparação com a programação estruturada
- Seus conceitos são de difícil compreensão

BENEFÍCIOS

- Mais fácil descrever o mundo real através dos objetos
- O encapsulamento facilita a manutenção do código
- Maior facilidade para reutilização de código

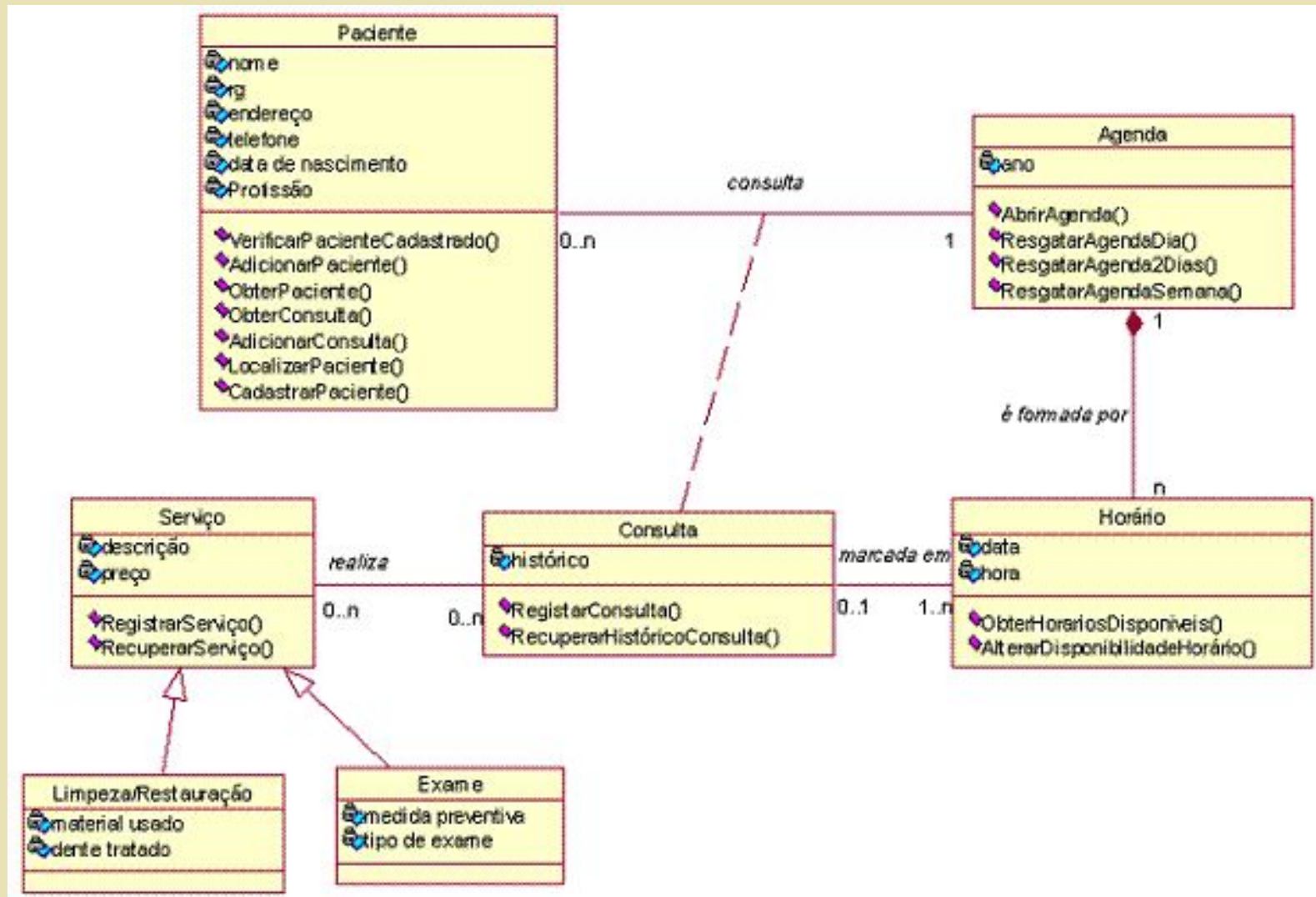
Conceitos Básicos

Motivação



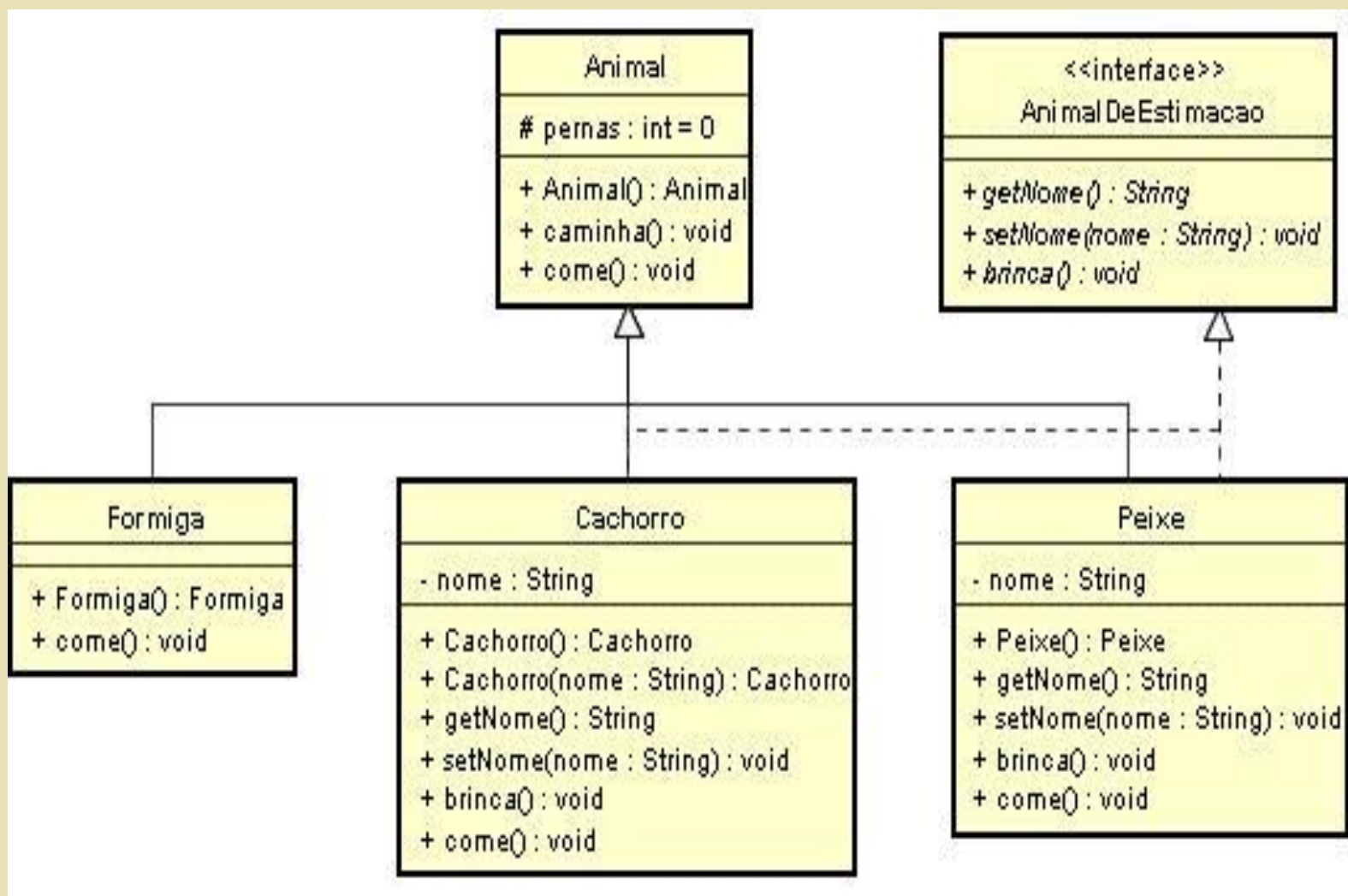
Conceitos Básicos

Motivação



Conceitos Básicos

Motivação





Introdução à POO

▣ Conceitos Básicos de OO

- ▣ Classe e Objeto
- ▣ Atributos
- ▣ Métodos e Operações
- ▣ Mensagem
- ▣ Encapsulamento
- ▣ Escopo de Classe
- ▣ Controle de Acesso a Membros (Visibilidade)

Conceitos Básicos

1 – Classe

◆ Definição:

- Representa um grupo de objetos semelhantes (classificação);
- É uma abstração de elementos do mundo real;
- É uma fábrica de objetos;
- Representa um tipo abstrato de dados;

◆ Exemplos:

Produto
<ul style="list-style-type: none">- codigo:int- nome:String- preco:doble
<ul style="list-style-type: none">+ alterarPreco(valor:doble):void+ getNome():String+ setNome(nome:String):void+ getCodigo():int+ setCodigo(codigo:int):void+ getPreco():doble

Carr
<ul style="list-style-type: none">Cor : StringPlaca: StringAno : Int
<ul style="list-style-type: none">ligar()acelerar()reduzir()

Aluno
<ul style="list-style-type: none">Matrícula : intNome: StringEndereço: String
<ul style="list-style-type: none">Matricular()CursarDisciplinas()

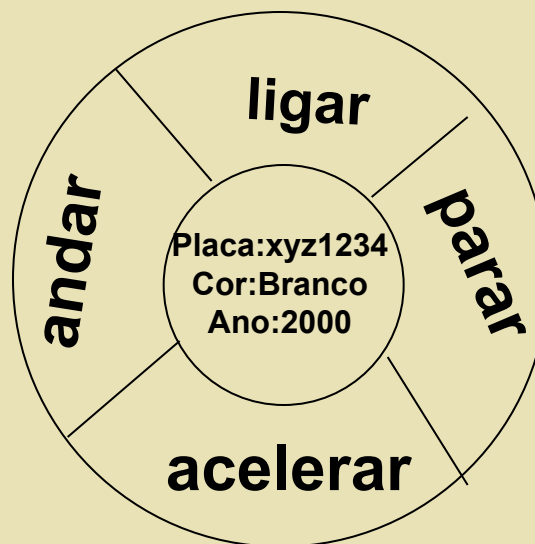
Conceitos Básicos

2 – Objetos

◆ Definição:

- É uma instância de uma classe
- Quando uma classe é instanciada, é alocado espaço na memória representando o objeto.

◆ Exemplo:

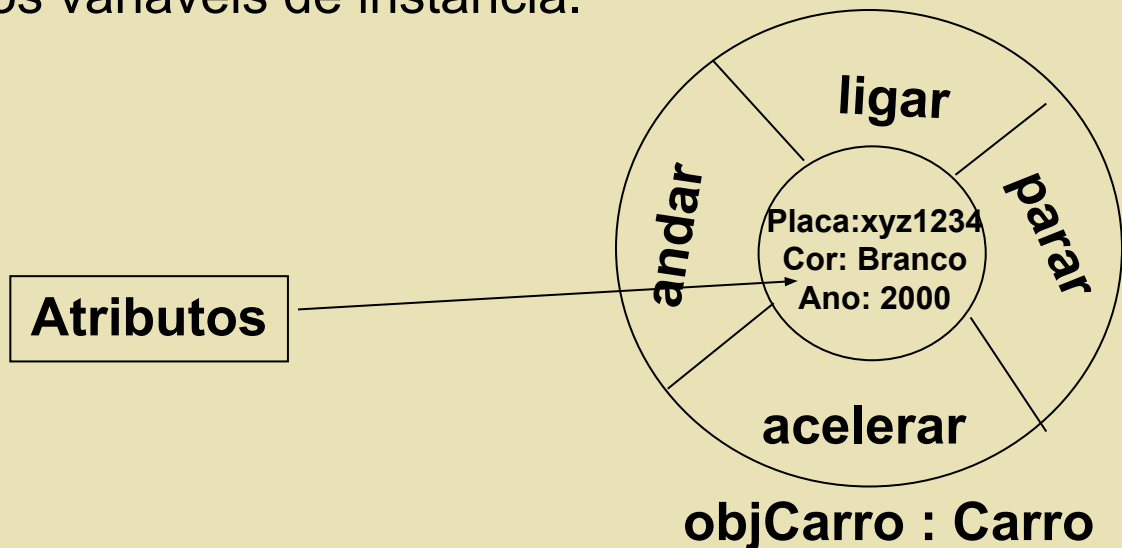


objCarro : Carro

Conceitos Básicos

3 – Atributos

- ◆ Utilizado para armazenar as características do objeto;
- ◆ Os objetos de uma classe possuem os mesmos atributos, porém os seus dados são normalmente diferentes;
- ◆ Contém os dados específicos de cada objeto;
- ◆ Os atributos de um objeto só devem ser manipulados pelo próprio objeto;
- ◆ Os atributos fisicamente na codificação são também chamados variáveis de instância.



Conceitos Básicos

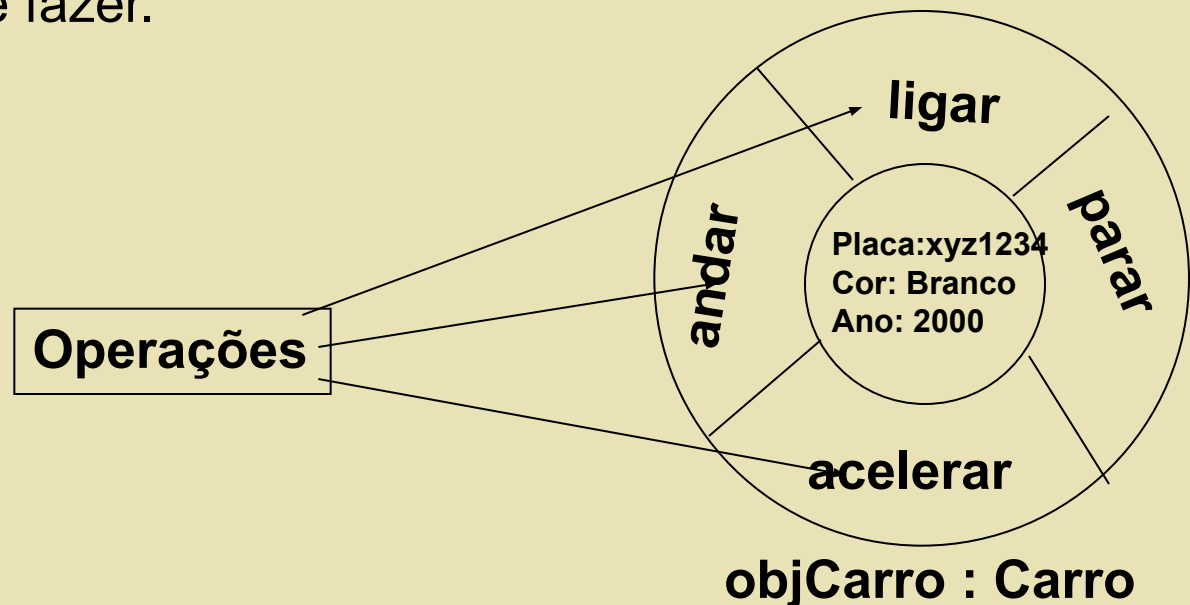
4 – Método e Operação

◆ Método

- É descrição do comportamento do Objeto.
- Representa o algoritmo de cada função desempenhada pelo objeto.

◆ Operação

- Corresponde ao nome do método, ou seja, apenas a interface do objeto, informando o que um objeto é capaz de fazer.



Conceitos Básicos

Exemplos de Objeto, Atributos e Operações

- ◆ Um objeto botão, pertencente à classe Botão:



Um exemplar da classe Botão

- ◆ **Atributos (características):**

- Cor, tamanho, posição no plano, ícone, etc.

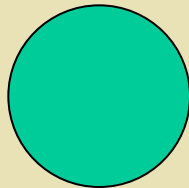
- ◆ **Operações:**

- Tornar-se visível ou invisível; mover-se; disparar um evento ao ser pressionado, etc.

Conceitos Básicos

Exemplos de Objeto, Atributos e Operações

- ◆ Um objeto círculo, pertencente à classe Círculo:



Um exemplar da classe Círculo

- ◆ **Atributos (características):**
 - Cor, tamanho, posição no plano(centro), raio.
- ◆ **Operações:**
 - Tornar-se visível ou invisível; informar sua área; informar seu perímetro, mover-se; disparar um evento ao ser pressionado, etc.



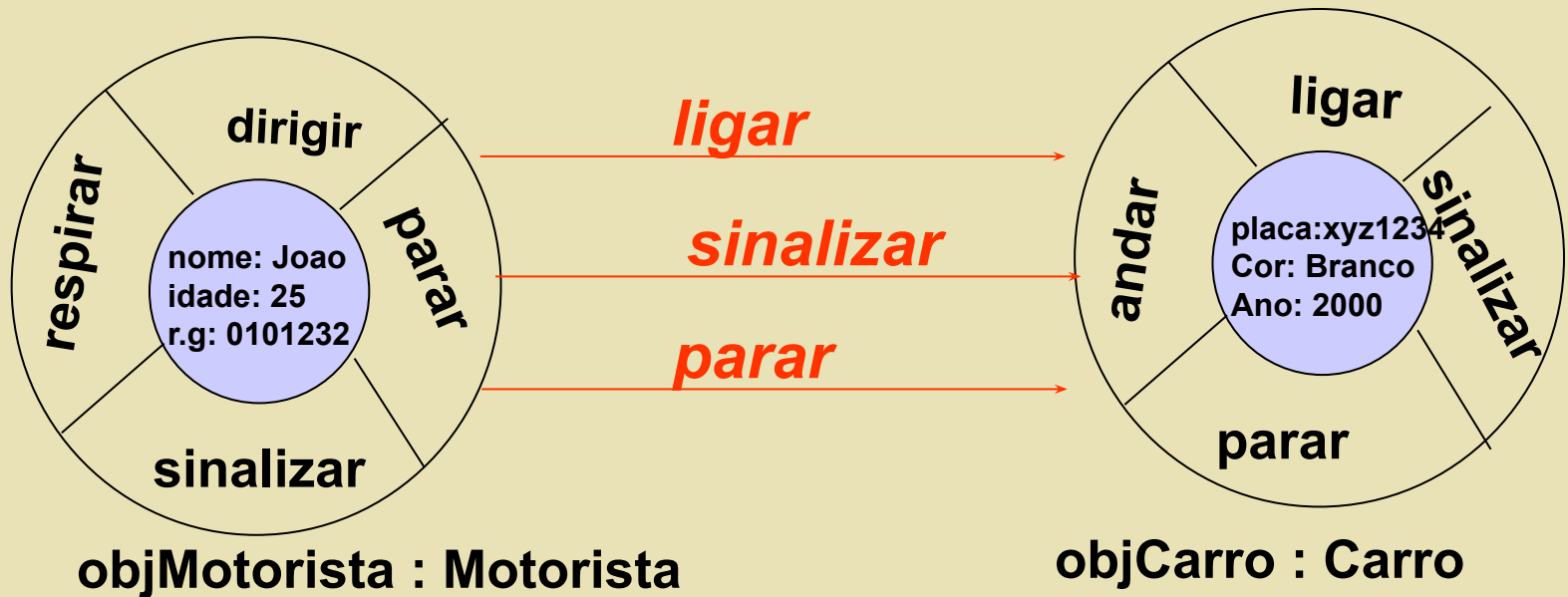
Conceitos Básicos

5 – Mensagem

- ◆ Mecanismo de comunicação entre objetos;
- ◆ As funcionalidades dos objetos são disponibilizadas através da sua interface; quando um objeto necessita de alguma funcionalidade de um outro, deve enviar-lhe uma mensagem;
- ◆ Após recebida a mensagem do objeto chamador, o objeto receptor da mensagem irá executar a funcionalidade solicitada.

Conceitos Básicos

5 – Mensagem (Exemplo)





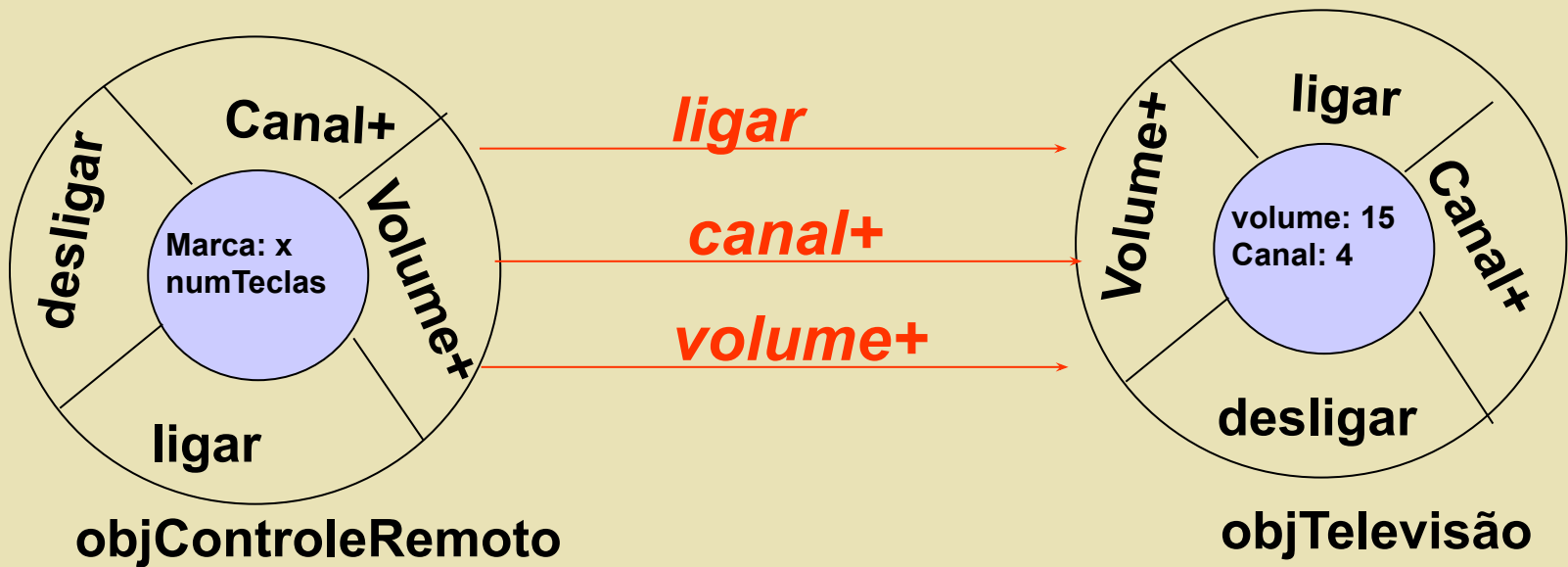
Conceitos Básicos

6 – Encapsulamento

- ◆ Princípio aplicado à programação orientada a objetos visando proteger os dados do objeto;
- ◆ Um objeto não deve permitir que nenhum outro objeto acesse seus dados diretamente;
- ◆ O relacionamento entre objetos deve ocorrer somente a partir de suas interfaces;
- ◆ Na implementação são usados métodos de acesso (discutido posteriormente) para se fazer referência aos dados do objeto.

Conceitos Básicos

6 – Encapsulamento (Exemplo)





Conceitos Básicos

7 – Escopo de Classe

- ◆ Atributos e métodos (membros) de uma classe pertencem ao escopo dessa classe;
- ◆ Dentro do escopo de uma classe os seus membros são imediatamente acessíveis para todos métodos dessa classe e podem ser referenciados diretamente pelo nome;
- ◆ Fora do escopo de uma classe, os membros da classe não podem ser referenciados diretamente pelo nome;
- ◆ Os membros de uma classe só podem ser referenciados através de um objeto da classe ou através do nome da própria classe
- ◆ Variáveis definidas em um método são conhecidas somente por esse método (são locais)



Conceitos Básicos

8 – Controle de Acesso a Membros (Visibilidade)

◆ Modificadores de acesso a membro

- Public (+) – visível e acessado pelos clientes da classe.
- Private (-) – somente visível e acessado na própria classe.
- Protected (#) - visível e acessado pelas subclasses*
- (omitido) - visível e acessado por todas as classes em um mesmo pacote*

* a ser visto posteriormente



Introdução à POO

▣ Conceitos Básicos de OO

- ▣ Atributos de Classe
- ▣ Métodos de Classe
- ▣ Métodos de Acesso
- ▣ Construtores



Conceitos Básicos

1 – Atributo de Classe

- ◆ A classe pode ser vista como um objeto especial capaz de gerar outros objetos, e como tal ela pode ter seus próprios atributos;
- ◆ Utilizados para armazenar as características da classe, ou seja, dados compartilhados por todos os objetos da classe.



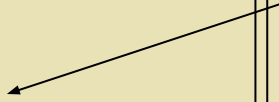
Conceitos Básicos

1 – Atributos de Classe (Exemplo)

```
public class Retângulo
{
    private float base;
    private float altura;
    private static int numLados = 4;

    public float area() {
        return base * altura;
    }
    public float perimetro() {
        return base * 2 + altura * 2;
    }
}
```

Atributo
de
classe





Conceitos Básicos

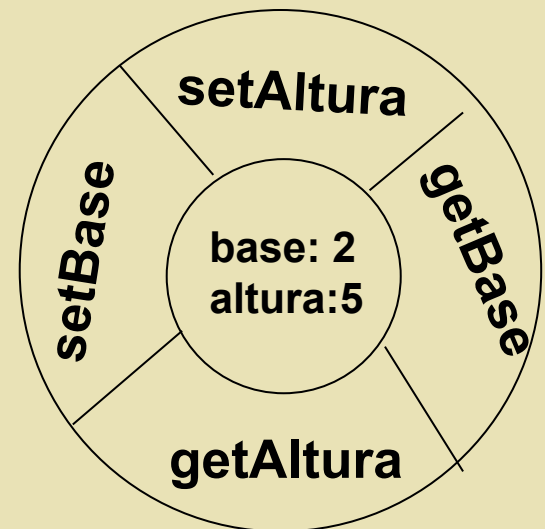
2 – Métodos de Classe

- ◆ Métodos que descrevem o comportamento da classe, e não dos objetos da classe.
- ◆ Só podem fazer referência a atributos de classe.
- ◆ Exemplos: criar objetos, destruir objetos.

Conceitos Básicos

3 – Métodos de Acesso

- São métodos definidos para garantir o encapsulamento dos dados do objeto.
- É um mecanismo prático de proteção dos atributos do objeto, onde qualquer acesso aos seus dados ocorre sempre através do envio de mensagem.
- Os métodos de acesso são:
set e get, associados ao nome do atributo ao qual fazem referência.



objRetângulo : Retângulo







Conceitos Básicos

3 – Métodos de Acesso (Exemplo)

Retangulo

  base : real

  altura : real

 setBase(num : real)
 getBase() : real
 setAltura(num : real)
 getAltura() : real
 area() : real
 perimetro() : real

```
public class Retangulo  
{
```

```
    private float base;  
    private float altura;
```

```
    public void setBase(float n)  
    { base = n; }  
    public float getBase()  
    { return base; }
```

```
    public void setAltura(float n)  
    { altura = n; }  
    public float getAltura()  
    {return altura; }  
}
```


Conceitos Básicos

4 – Construtores

- ◆ São métodos especiais que são executados imediatamente após a instanciação da classe (criação de objeto).
- ◆ Os construtores devem ter o mesmo nome da classe, e não apresentam parâmetro de retorno, embora possam receber parâmetros.
- ◆ Os construtores são chamados automaticamente quando um objeto é criado;
- ◆ Uma classe pode definir vários métodos construtores.

construtores

```
public class Numero {  
    int valor;  
    public Numero() {  
        valor = 0;  
    }  
    public Numero(int x) {  
        valor = x;  
    }  
}
```

Exemplo

- ◆ Criar uma classe chamada Ponto contendo os atributos x e y . A seguir criar diversos objetos ponto em um objeto Plano Cartesiano;



Exemplo

Criação da Classe, Atributos e Métodos

```
public class Ponto {  
    private int x, y;  
    public Ponto (int a, int b) { x = a; y = b; }  
    public void moverHor(int i) { x = x + i; }  
    public int getx() { return x; }  
    public int gety() { return y; }  
    public double distancia(Ponto p) {  
        double deltax = p.getx() - x;  
        double deltay = p.gety() - y;  
        return Math.sqrt(Math.pow(deltax,2) + Math.pow(deltay,2));  
    }  
}
```

Construtor

Métodos

Exemplo

Criação da Classe PlanoCartesiano e do Objeto Ponto

```
public class PlanoCartesiano {  
    public static void main(String args[])  
    {  
        int x1=2, y1=1;  
        Ponto p1 = new Ponto(x1, y1);  
        Ponto p2 = new Ponto(10,5);  
        System.out.println(p1.distancia(p2));  
    }  
}
```

instanciação

envio de mensagem

Introdução à POO

▣ Conceito Gerais

- ▣ THIS
- ▣ SUPER
- ▣ Herança






Conceito de THIS

- ◆ Mecanismo usado para fazer referência ao objeto que está sendo executado num determinado momento.
- ◆ Conceito bastante utilizado em programação OO para indicar o objeto de referência, tornando a programação mais clara.
- ◆ É utilizado para se ter acesso aos atributos e métodos do objeto de referência ou ao próprio objeto.
- ◆ Evita conflitos entre variáveis locais e atributos do objeto de referência;

Exemplo de THIS (1)



```
public class Numero {  
    int valor;  
    public setValor (int valor)  
    { this.valor = valor; }  
  
    public Numero dobro()  
    {   valor = valor *2;  
        return this;  
    }  
}
```

```
public class Teste {  
    int x,y;  
    public void inicia(int x1,y1)  
    { this.x=x1;  
      this.y=y1; } }
```

```
public Numero dobro() {  
    num = new Numero;  
    num.setValor(valor*2);  
    return num;  
} // tem que instanciar
```

Com geração de um
novo objeto

Exemplo de THIS (2)

Implementar o método soma para adicionar um valor inteiro ao valor atual de uma instância da classe Número.

```
public Numero soma (int n) {  
    // adiciona n ao valor atual do objeto  
    valor = valor + n;  
    // retorna o próprio objeto de referência  
    return this;  
}
```

Retorna um objeto Numero

Conceito de SUPER

- ◆ Mecanismo que faz referência à superclasse de um objeto visando a execução de um método.
- ◆ Também pode referir-se diretamente ao(s) construtor(es) da superclasse.

<u>Bicicleta</u>
<u>cor</u>
<u>fabricante</u>
<u>andar()</u> <u>//andar p/ frente</u>



<u>Bicicleta_Circo</u>
<u>cor</u>
<u>fabricante</u>
<u>super.andar()</u> <u>this.andar()</u>

Exemplo de SUPER (1)

**Conceito de
SUPER
referindo-se
ao método
construtor da
superclasse**

```
public class MeuFrame extends JFrame {  
  
    public MeuFrame() {  
        super("Título do Frame");  
        this.setSize(200,200);  
        this.show();  
    }  
}
```

```
public class MenuFrame extends JFrame {  
    public MenuFrame() {  
        this.setTitle("Título do Frame");  
        this.setSize(200,200);  
        this.show();    }  
    public void setTitle(String msg)  
    { super.setTitle(msg+"Versão 1.0"); }  
}
```

**Observações:
Super
refere-se a
JFrame e
This refere-se
a MenuFrame**

Exemplo de SUPER (2)

```
public class Ponto3D extends Ponto {  
    int z;  
    public Ponto3D(int a,b,c) {  
        super(a,b); //chama o construtor da superclasse  
        this.z=c;  
    }  
    public static void main(String args[] {  
        Ponto3D p = new Ponto3D (10,20,30);  
        System.out.println("x="+p.x+"y="+p.y+"z="+p.z)  
    }  
}
```



Conceito de Herança

- ◆ Um dos recursos mais fundamentais da OO, pois viabiliza a criação de hierarquia de classes.
- ◆ Refere-se à capacidade de uma classe se especializar, i.e., novas classes podem ser definidas em termos das classes existentes através da herança de classe;
- ◆ Cria-se inicialmente uma classe genérica (superclasse / classe pai-mãe / classe base) que define as características e comportamentos comuns a um conjunto de itens inter-relacionados.
- ◆ A partir dela pode-se derivar outras classes mais específicas (subclasses / classes filhas / classes derivadas) que herdam suas características e comportamentos.



Conceito de Herança

- ◆ As novas classes derivadas além de herdarem as características e comportamentos (atributos e operações) da classe base, também podem ao mesmo tempo acrescentar novos recursos mais específicos.
- ◆ Os objetos que são instanciados de uma subclasse conterão todos os atributos e operações definidas por ela e por sua superclasse, sendo capazes de executar todas as operações definidas por esta subclasse e por seus “ancestrais”;
- ◆ Garante e permite o reuso do comportamento de uma classe na definição de novas classes.
- ◆ Herança Simples X Herança Múltipla

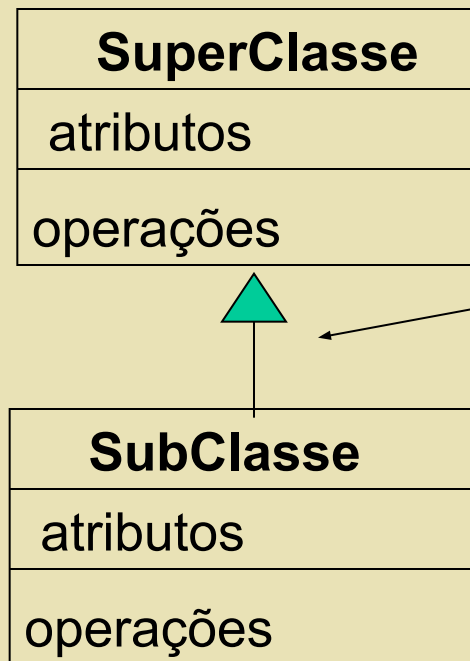


Conceito de Herança

- ◆ Comportamento: os objetos da subclasse comportam-se como os objetos da superclasse.
- ◆ Substituição: os objetos da subclasse podem ser usados no lugar de objetos da superclasse.
- ◆ Reuso de Código: a descrição da superclasse pode ser usada para definir a subclasse.
- ◆ Extensibilidade: algumas operações da superclasse podem ser redefinidas na subclasse.
- ◆ Subclasse extends Superclasse.
- ◆ Atributos e métodos privados que são herdados não podem ser acessados diretamente.
- ◆ Qualificador protected: visibilidade restrita à superclasse e às subclasses.
- ◆ Construtores não são herdados.

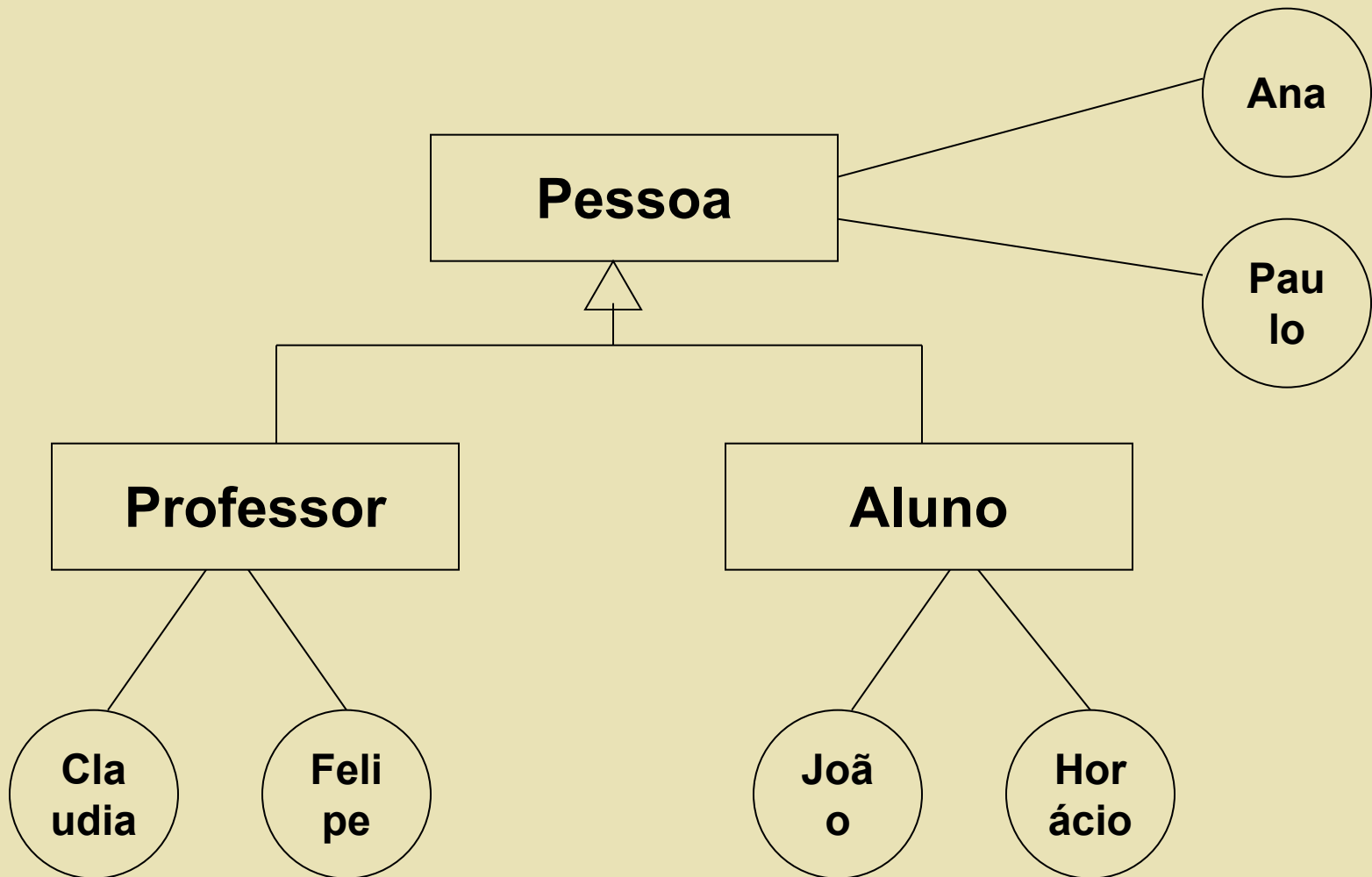
Conceito de Herança

- ♦ A notação UML para representação do relacionamento de herança é a seguinte:



**representação
de herança
(especialização
e/ou
generalização)**

Exemplo de Herança (1)





Exemplo de Herança (2)

```
public class Ponto {  
    int x,y;  
    public Ponto(int a, b) {  
        this.x=a;  
        this.y=b;    }  
}
```

```
public class Ponto3D extends Ponto {  
    int z;  
    public Ponto3D (int a, b, c) {  
        this.x=a;  
        this.y=b;  
        this.z=c;    }  
    public Ponto3D();  
    { this(-1,-1,-1); } //chamada ao próprio construtor  
}
```

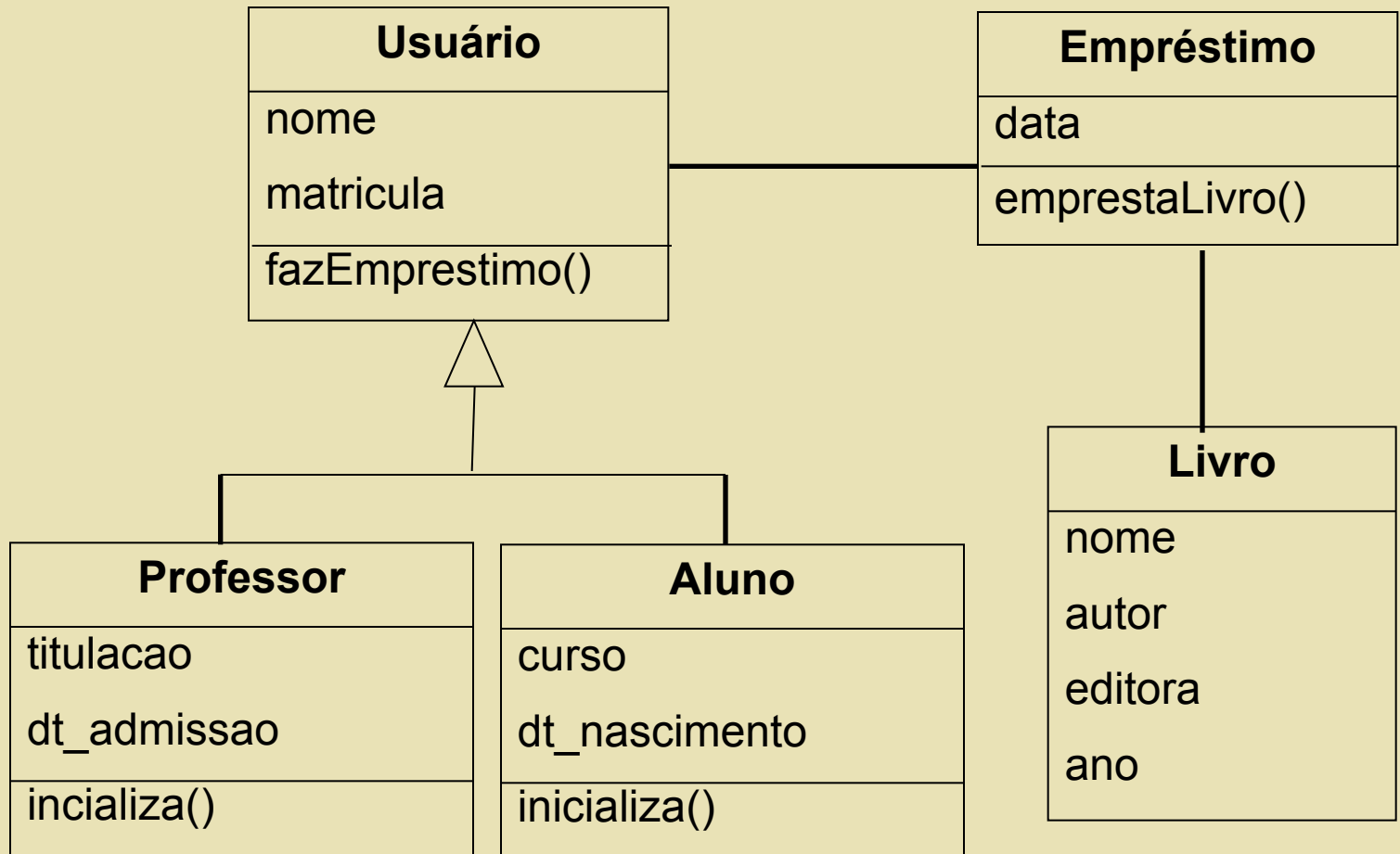


Exemplo de Herança (3)

```
public class Employee {  
    private String name;  
    private Date hireDate;  
    private Date dateofBirth;  
    private String jobTitle;  
}
```

```
public class Manager extends Employee {  
    private String department;  
    private Employee[] subordinates;  
}
```


Exercícios de Herança





Exercícios de Herança

Implementação da Classe Abstrata Usuário

```
public abstract class Usuario {  
    private String nome;  
    private String matricula;  
    private Emprestimo refEmprestimo;  
  
    public void setNome(String nm)  
    {this.nome=nm;}  
  
    public void setMatricula(String matr)  
    {this.matricula=matr;}  
  
    public void setRefEmprestimo(Emprestimo e)  
    {this.refEmprestimo=e;}  
}
```



Exercícios de Herança

Implementação da Classe Abstrata Usuário (continuação)

```
public String getNome()  
{return this.nome;}
```

```
public String getMatricula()  
{return this.matricula;}
```

```
public Emprestimo getRefEmprestimo()  
{return this.refEmprestimo;}
```

```
public abstract void fazEmprestimo();
```

```
}
```



Exercícios de Herança

Implementação da Classe Professor

```
public class Professor extends Usuario {
```

```
    String titulacao;  
    String dt_admissao;
```

```
    public void setTitulacao(String tit)  
    {this.titulacao=tit;}
```

```
    public void setDt_admissao(String dt)  
    {this.dt_admissao=dt;}
```

```
    public String getTitulacao()  
    {return this.titulacao;}
```

```
    public String getDt_admissao()  
    {return this.dt_admissao;}
```




Exercícios de Herança

Implementação da Classe Professor (continuação)

```
public void inicializa()
{
    this.setNome("João");
    this.setMatricula("12345");
    this.setTitulacao("Mestrado");
    this.setDt_admissao("09/09/2003");
    this.fazEmprestimo();
}
```



Exercícios de Herança

Implementação da Classe Professor (continuação)

```
public void fazEmprestimo()  
{  
    Livro objLivro=new Livro();  
    objLivro.setNome("Java em 2 Dias");  
    objLivro.setAutor("Pedro Paulo");  
    objLivro.setEditora("Makron Books");  
    objLivro.setAno(2002);  
    Emprestimo objEm = new Emprestimo();  
    objEm.setRefLivro(objLivro);  
    objEm.setData("10/11/2003");  
    this.setRefEmprestimo(objEm);  
}
```



Exercícios de Herança

Classe Aluno

```
public class Aluno extends Usuario {  
    String curso;  
    String dt_nasc;  
  
    public void setCurso(String cur)  
    {this.curso=cur;}  
    public void setDt_nasc(String dt)  
    {this.dt_nasc=dt;}  
    public String getCurso()  
    {return this.curso;}  
    public String getDt_nasc()  
    {return this.dt_nasc;}  
}
```



Exercícios de Herança

Implementação da Classe Aluno (continuação)

```
public void inicializa()
{
    this.setNome("Roberto");
    this.setMatricula("54321");
    this.setCurso("Informática");
    this.setDt_nasc("02/05/1978");
    this.fazEmprestimo();
}
```


Exercícios de Herança

Implementação da Classe Aluno(continuação)

```
public void fazEmprestimo()
{
    Livro objLivro=new Livro();
    objLivro.setNome("Java em 21 Dias");
    objLivro.setAutor("Paulo Pedro");
    objLivro.setEditora("Makron Books");
    objLivro.setAno(2003);
    Emprestimo objEm = new Emprestimo();
    objEm.setRefLivro(objLivro);
    objEm.setData("11/11/2003");
    this.setRefEmprestimo(objEm);
}
```



Exercícios de Herança

Implementação da Classe Livro

```
public class Livro {  
  
    String nome;  
    String autor;  
    String editora;  
    int ano;  
  
    public void setNome (String n)  
    { nome = n; }  
    public String getNome()  
    { return nome; }  
    public void setAutor(String a)  
    { autor=a;}  
}
```



Exercícios de Herança

Implementação da Classe Livro (continuação)

```
public String getAutor()
{return autor; }
public void setEditora(String e)
{ editora =e; }
public String getEditora()
{ return editora;}
public void setAno(int a)
{ ano=a; }
public int getAno()
{ return ano; }
}
```



Exercícios de Herança

Implementação da Classe Empréstimo

```
public class Emprestimo{  
    String data;  
    Livro refLivro;  
  
    public void setData(String dt)  
    { data =dt;    }  
    public String getData()  
    { return data;}  
    public void setRefLivro(Livro lv)  
    { refLivro =lv;    }  
    public Livro getRefLivro()  
    { return refLivro;}  
}
```




Exercícios de Herança

Implementação da Classe Teste

```
public class Teste{  
  
    public static void main(String args[]) {  
        Professor p=new Professor();  
        p.inicializa();  
        Emprestimo pe = p.getRefEmprestimo();  
        Livro pl = pe.getRefLivro();  
        Aluno a=new Aluno();  
        a.inicializa();  
        Emprestimo ae = p.getRefEmprestimo();  
        Livro al = ae.getRefLivro();  
    }  
}
```



Exercícios de Herança

Implementação da Classe Teste (continuação)

```
System.out.println("RELAÇÃO DOS EMPRÉSTIMOS DE  
LIVROS FEITOS: ");  
    System.out.println("\nEMPRÉSTIMO FEITO POR  
PROFESSOR: ");  
        System.out.println("Nome do Livro.....: "+pl.getNome());  
        System.out.println("Autor.....: "+pl.getAutor());  
        System.out.println("Editora.....: "+pl.getEditora());  
        System.out.println("Ano.....: "+pl.getAno());  
        System.out.println("Nome do Professor.: "+p.getNome());  
        System.out.println("Matricula.....: "+p.getMatricula());  
        System.out.println("Titulação.....: "+p.getTitulacao());  
        System.out.println("Data de admissao..  
"+p.getDt_admissao());  
        System.out.println("Data do empréstimo: "+pe.getData());
```

Exercícios de Herança

Implementação da Classe Teste (continuação)

```
System.out.println("\nEMPRÉSTIMO FEITO POR ALUNO: ");
System.out.println("Nome do Livro.....: "+al.getNome());
System.out.println("Autor.....: "+al.getAutor());
System.out.println("Editora.....: "+al.getEditora());
System.out.println("Ano.....: "+al.getAno());
System.out.println("Nome do Professor.: "+a.getNome());
System.out.println("Matricula.....: "+a.getMatricula());
System.out.println("Titulação.....: "+a.getCurso());
System.out.println("Data de admissao..: "+a.getDt_nasc());
System.out.println("Data do empréstimo: "+ae.getData());
}
}
```



Exercícios de Herança

RESULTADOS

RELAÇÃO DOS EMPRÉSTIMOS DE LIVROS FEITOS:

EMPRÉSTIMO FEITO POR PROFESSOR:

Nome do Livro.....: Java em 2 Dias

Autor.....: Pedro Paulo

Editora.....: Makron Books

Ano.....: 2002

Nome do Professor.: João

Matricula.....: 12345

Titulação.....: Mestrado

Data de admissão..: 09/09/2003

Data do empréstimo: 10/11/2003

Exercícios de Herança

RESULTADOS (CONTINUAÇÃO)

EMPRÉSTIMO FEITO POR ALUNO:

Nome do Livro.....: Java em 2 Dias

Autor.....: Pedro Paulo

Editora.....: Makron Books

Ano.....: 2002

Nome do Professor.: Roberto

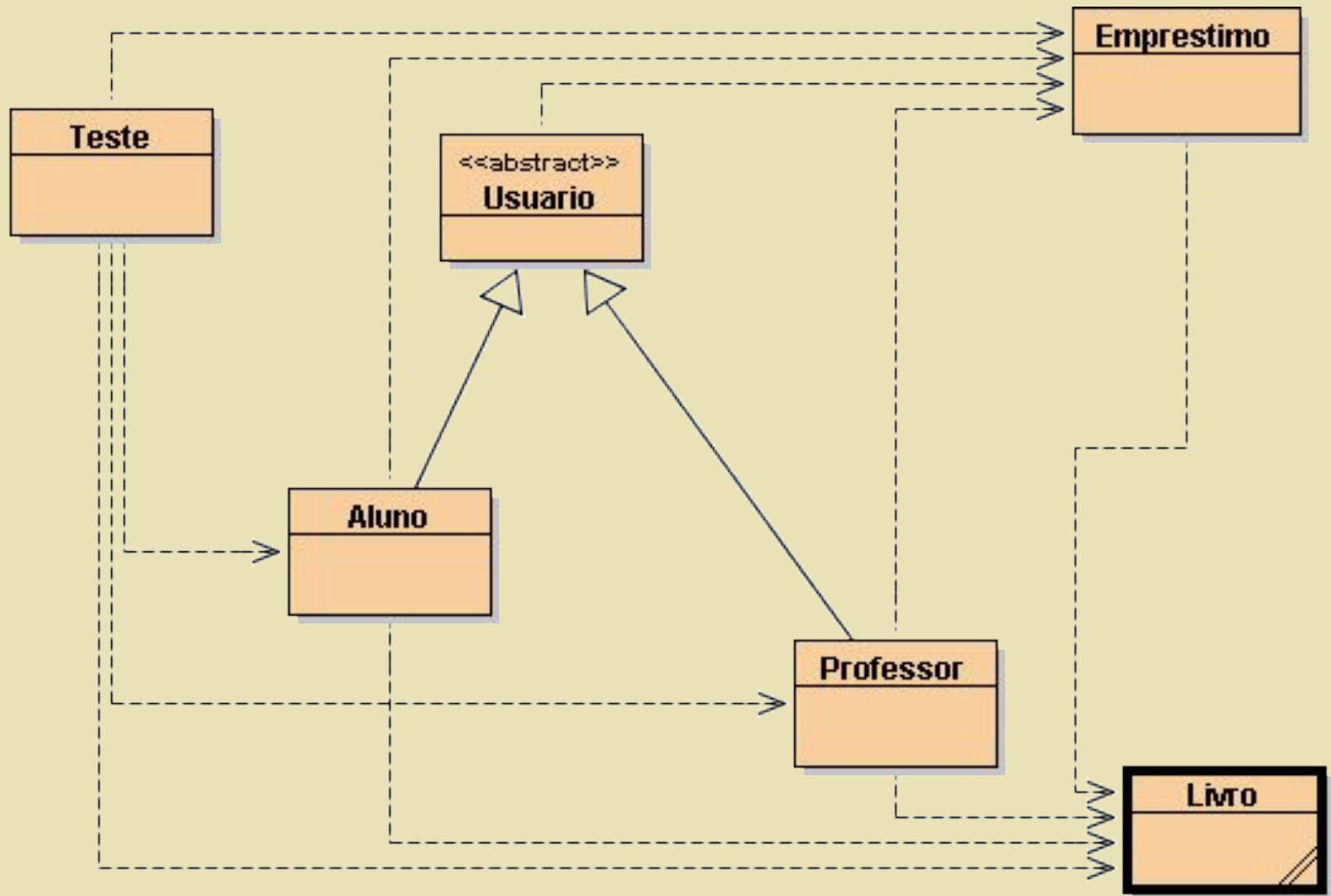
Matricula.....: 54321

Titulação.....: Informática

Data de admissão..: 02/05/1978

Data do empréstimo: 10/11/2003

Exercícios de Herança



Introdução à POO

▣ Conceito de Polimorfismo

- ▣ Polimorfismo Sobrecarga
- ▣ Polimorfismo Sobreposição





Conceito de Polimorfismo

- ◆ Nas linguagens OO, a interpretação de uma mensagem é da responsabilidade do objeto que a recebe. Isto permite que uma mesma mensagem possa ser interpretada de forma diferente por diferentes objetos.
- ◆ As operações que possuem esta característica são chamadas polimórficas.
- ◆ A implementação do polimorfismo se dá através de um mecanismo chamado *late-binding*, o que significa que a definição do método que será executado a partir de uma mensagem enviada a um objeto só irá ocorrer em tempo de execução, quando de fato for identificado o objeto que deverá interpretar a mensagem.

Exemplo de Polimorfismo

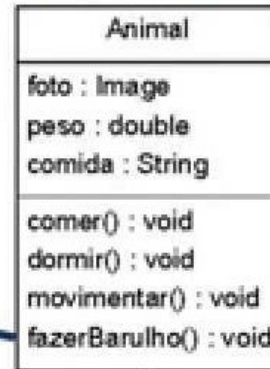
Polimorfismo

Comportamento Diferente

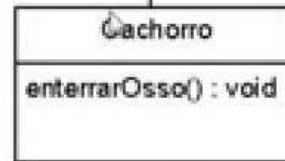
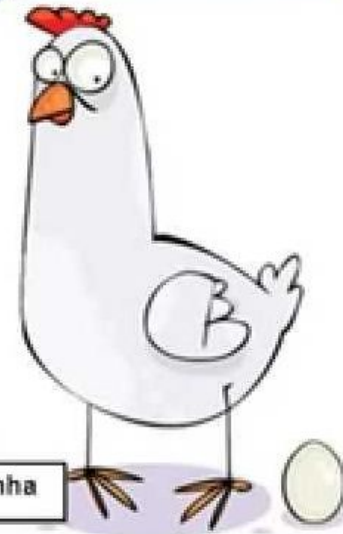
Au, Au !



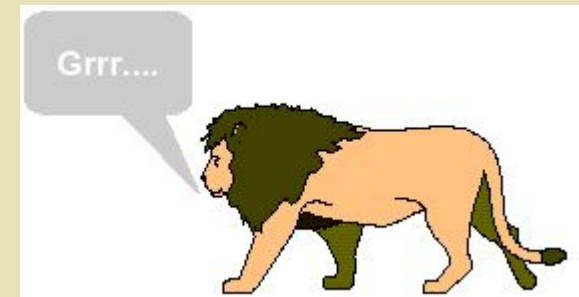
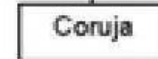
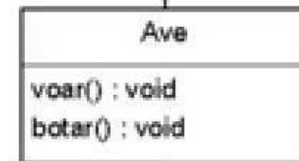
Superclasse



Có, Có !



Subclasses





Conceito de Polimorfismo

- ◆ A associação em tempo de execução de uma solicitação a um objeto e a uma de suas operações é conhecida como ligação dinâmica (dynamic binding).
- ◆ A ligação dinâmica permite substituir objetos que tenham interfaces idênticas uns pelos outros. Esta capacidade de substituição é conhecida como Polimorfismo.
- ◆ Polimorfismo simplifica as definições dos clientes, desacopla objetos entre si, permitindo que eles variem seus inter-relacionamentos em tempo de execução.



Conceito de Polimorfismo

- ◆ Uma mesma mensagem é enviada a diferentes objetos que ao receberem esta mensagem podem responder de forma diferente à mesma mensagem.
- ◆ Um mesmo método pode ser implementado de diversas formas em classes diferentes.
- ◆ Existem 2 tipos de polimorfismo:
 - ◆ Polimorfismo do Tipo Sobrecarga e;
 - ◆ Polimorfismo do Tipo Sobreposição.



Polimorfismo Sobrecarga

- ◆ A forma mais simples de polimorfismo oferecido pela linguagem Java é a sobrecarga de métodos (*method overload*), ou seja, é a possibilidade de existirem numa mesma classe vários métodos com o mesmo nome.
- ◆ Para que estes métodos de mesmo nome possam ser distinguidos, eles devem possuir uma assinatura diferente.



Polimorfismo Sobrecarga

```
public class Ponto {  
    int x, y;  
    Ponto (int a, int b) {  
        this.x = a;  
        this.y = b;}  
    Ponto ( ) {  
        x = -1;  
        y = -1; }  
}
```

```
public class CriaPontoAlt {  
    public static void main (String args[ ]) {  
        Ponto p = new Ponto( );  
        System.out.println( "x = " +p.x + " y = " + p.y);  
        Ponto p2 = new Ponto(3,3);  
        System.out.println( "x = " +p2.x + " y = " + p2.y);  
    }  
}
```



Polimorfismo Sobreposição

- ◆ Um método de uma subclasse se sobrepõe (override) a um método de uma superclasse quando o método da subclasse tem a mesma assinatura , ou seja:
 - retorna o mesmo tipo
 - tem o mesmo número e os mesmo tipos de parâmetros
 - tem o mesmo nome
- ◆ Quando o método sobreposto é chamado de dentro da subclasse, a versão executada é sempre aquela definida na subclasse.

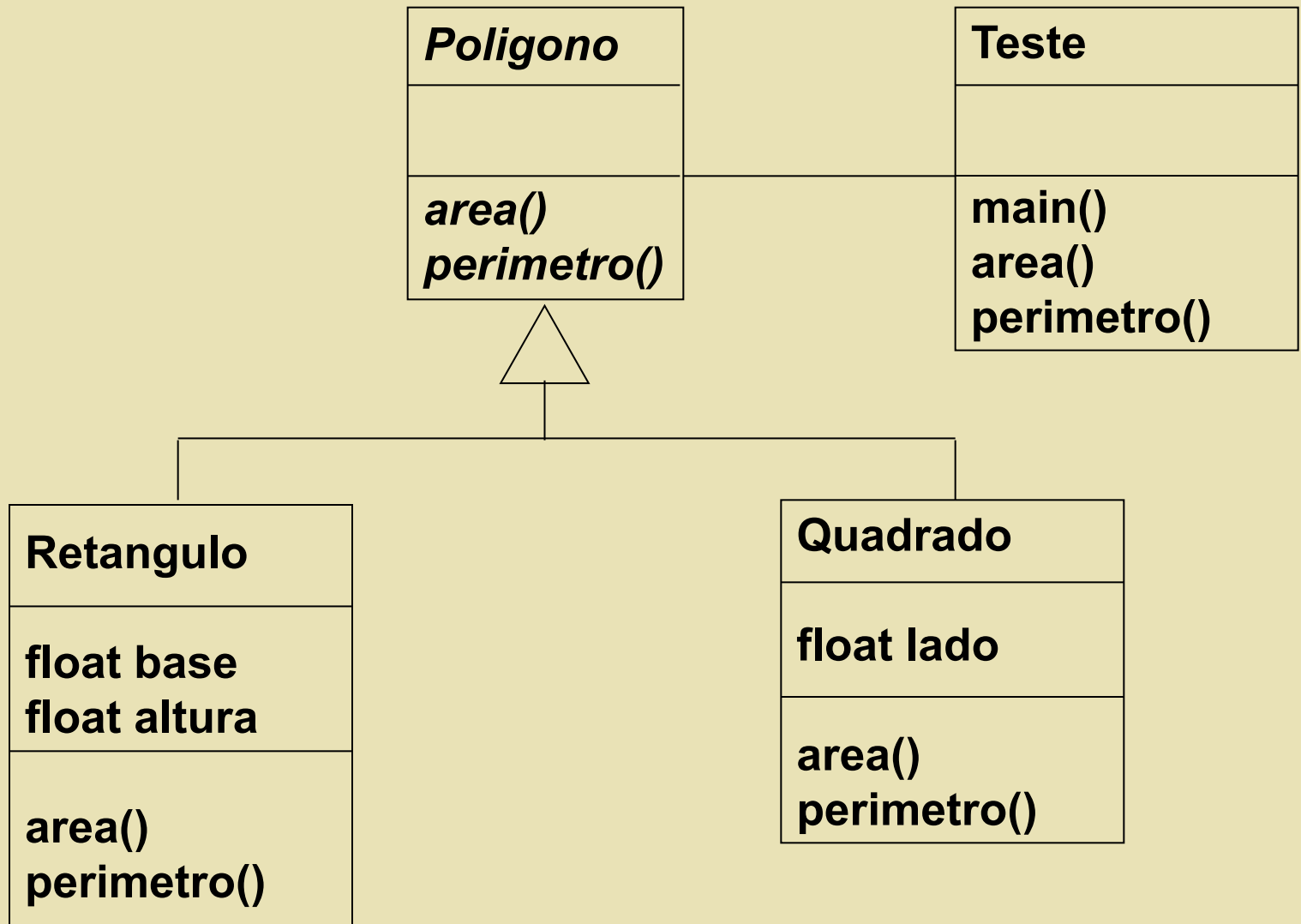


Polimorfismo Sobreposição

```
public class Computador {  
    private boolean ligado = true;  
    public void Desligar()  
        { ligado = false; }  
}
```

```
public class ComputadorSeguro extends Computador {  
    private boolean executando = true;  
    public void Desligar( ) {  
        if ( executando )  
            {System.out.println("Programas rodando. Não desligue!");}  
        else  
            { ligado = false;}  
    }  
}
```

Exemplo de Polimorfismo



Exemplo de Polimorfismo

Classe Abstrata

```
public abstract class Poligono {  
    public abstract float area();  
    public abstract float perimetro();  
}
```

métodos abstratos





Exercícios de Polimorfismo

```
public class Retangulo extends Poligono {  
    private float base;  
    private float altura;  
    public void setBase(float b)  
        { this.base = b; }  
    public void setAltura(float a)  
        { this.altura = a; }  
    public float getBase()  
        { return this.base; }  
    public float getAltura()  
        { return this.altura; }  
    public float area()  
        { return base*altura; }  
    public float perimetro()  
        { return 2*base+2*altura; }  
}
```



Exercícios de Polimorfismo

```
public class Quadrado extends Poligono {  
    private float lado;  
  
    public void setLado(float l)  
        { this.lado = l; }  
  
    public float getLado()  
        { return this.lado; }  
  
    public float area()  
        { return lado*lado; }  
  
    public float perimetro()  
        { return lado+lado; }  
}
```

Exercícios de Polimorfismo

```
public static void main(String args[]) {  
    if (args[0].equalsIgnoreCase(args[0])) {  
        Quadrado q = new Quadrado();  
        q.setLado(Float.parseFloat(args[1]));  
        System.out.println("Area = "+q.area());  
        System.out.println("Perimetro =" +q.perimetro());  
    }  
    else {  
        Retangulo r = new Retangulo();  
        r.setBase(Float.parseFloat(args[1]));  
        r.setAltura(Float.parseFloat(args[2]));  
        System.out.println("Area = "+r.area());  
        System.out.println("Perimetro =" +r.perimetro());  
    }  
}}
```


Introdução à POO

- Classe Abstrata
- Interface



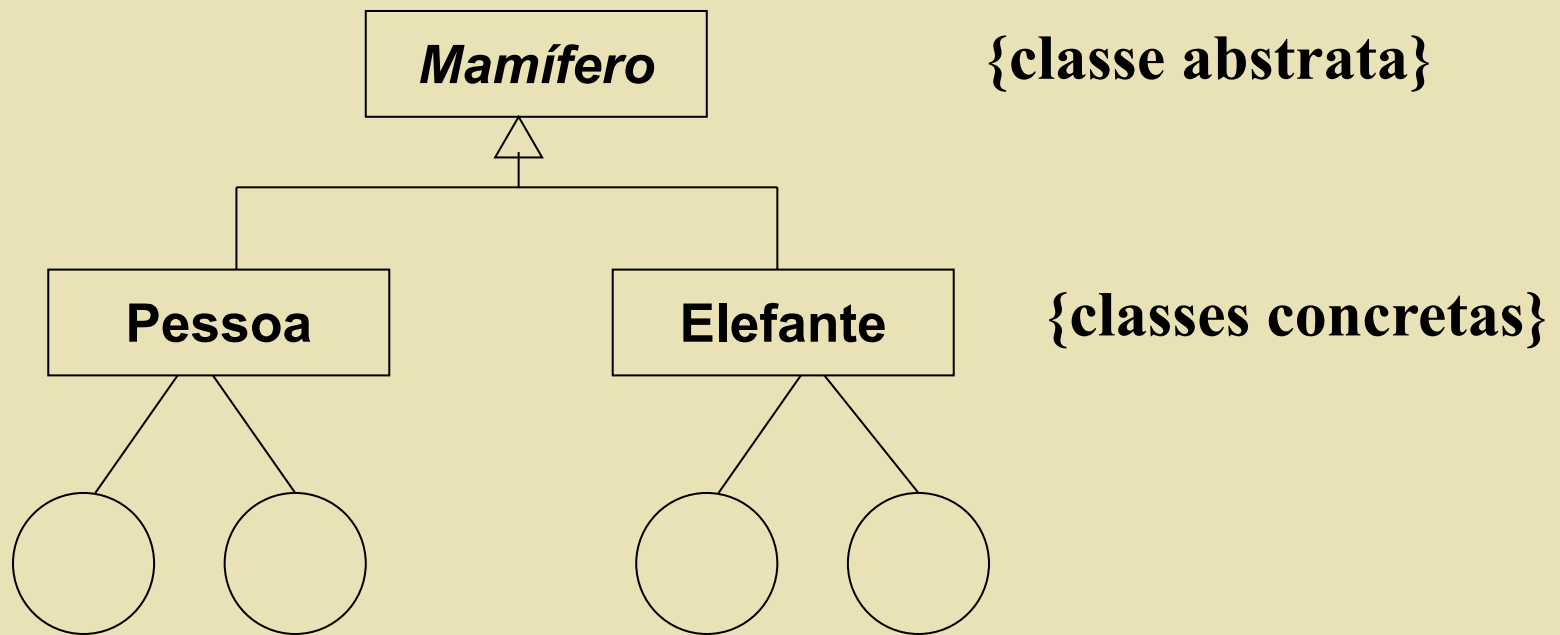


Conceito de Classe Abstrata

- ◆ Modela a semântica do mundo real, porém não é utilizada para gerar objetos (não pode ser instanciada).
- ◆ Sua principal finalidade é definir uma interface comum para suas subclasses.
- ◆ Ela postergará parte ou toda sua implementação para operações definidas em subclasses.
- ◆ As operações que uma classe abstrata declara, mas não implementa são chamadas operações abstratas.

Exemplo de Classe Abstrata

- ◆ As classes que não são abstratas são chamadas classes concretas.





Conceito de Interface

- ◆ Elimina a necessidade de herança múltipla.
- ◆ É semelhante a uma classe abstrata uma vez que não se destina a ser instanciada.
- ◆ Sua intenção é oferecer uma interface que envolva operações comuns a diferentes classes ou hierarquias de classes.
- ◆ Define um conjunto de operações que deverão ser implementadas por quem utilizá-la.
- ◆ Diz-se que uma classe pode implementar uma interface, ou seja, implementa todas as operações definidas na interface.



Conceito de Interface

- ◆ Corresponde a uma definição abstrata que pode ser implementada por determinadas classes (ex.: interface de um relógio, calculadora, etc.)
- ◆ Em qualquer lugar que uma interface for requerida, pode-se utilizar qualquer objeto que implemente a interface;



Conceito de Interface

- ◆ Definições:
 - dispositivo que permite a interação entre objetos distintos;
 - mecanismo que define um protocolo de comportamento que pode ser implementado por qualquer classe do sistema.
- ◆ A interface funciona como um novo tipo, o que permite que ela seja utilizada em lugar do tipo do objeto que a implementa.

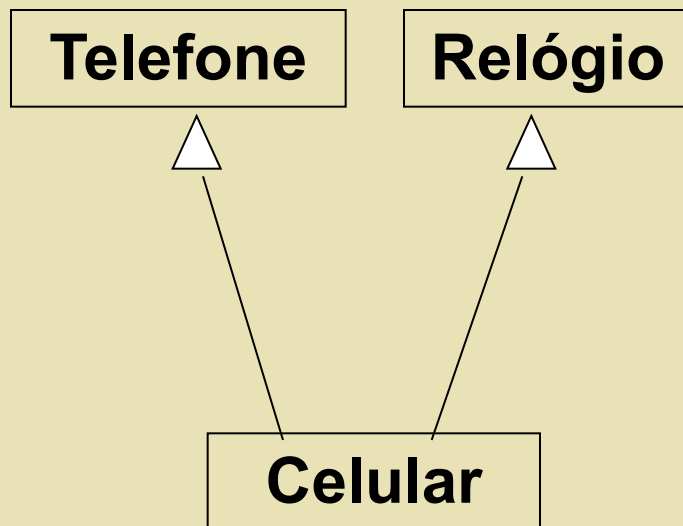


Características das Interfaces

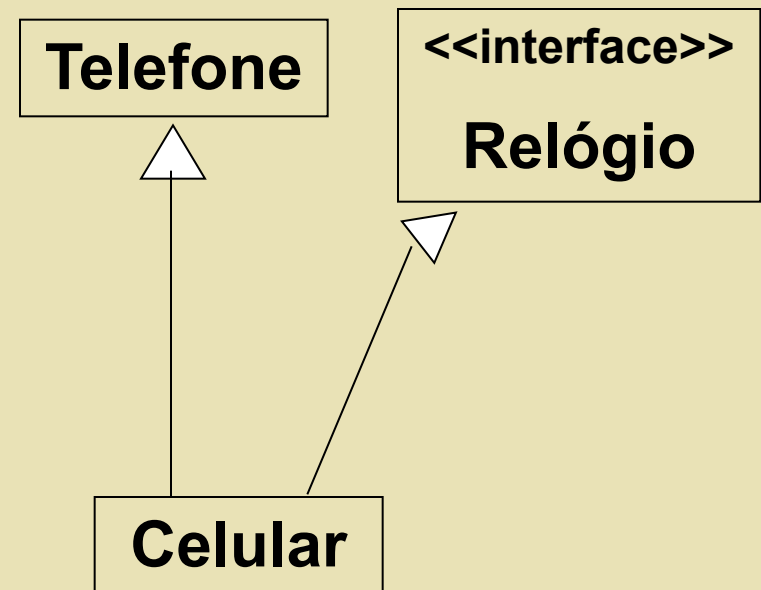
- ◆ Capturam similaridades entre classes independentes, sem forçar relacionamentos;
- ◆ Declaram métodos que uma ou mais classes deverão implementar;
- ◆ Revelam a interface de um objeto sem revelar a classe do objeto.

Solução Através de Interface

Herança Múltipla (C++)



Interface (Java)



Relógio não é uma classe e sim uma INTERFACE.

Solução Através de Interface

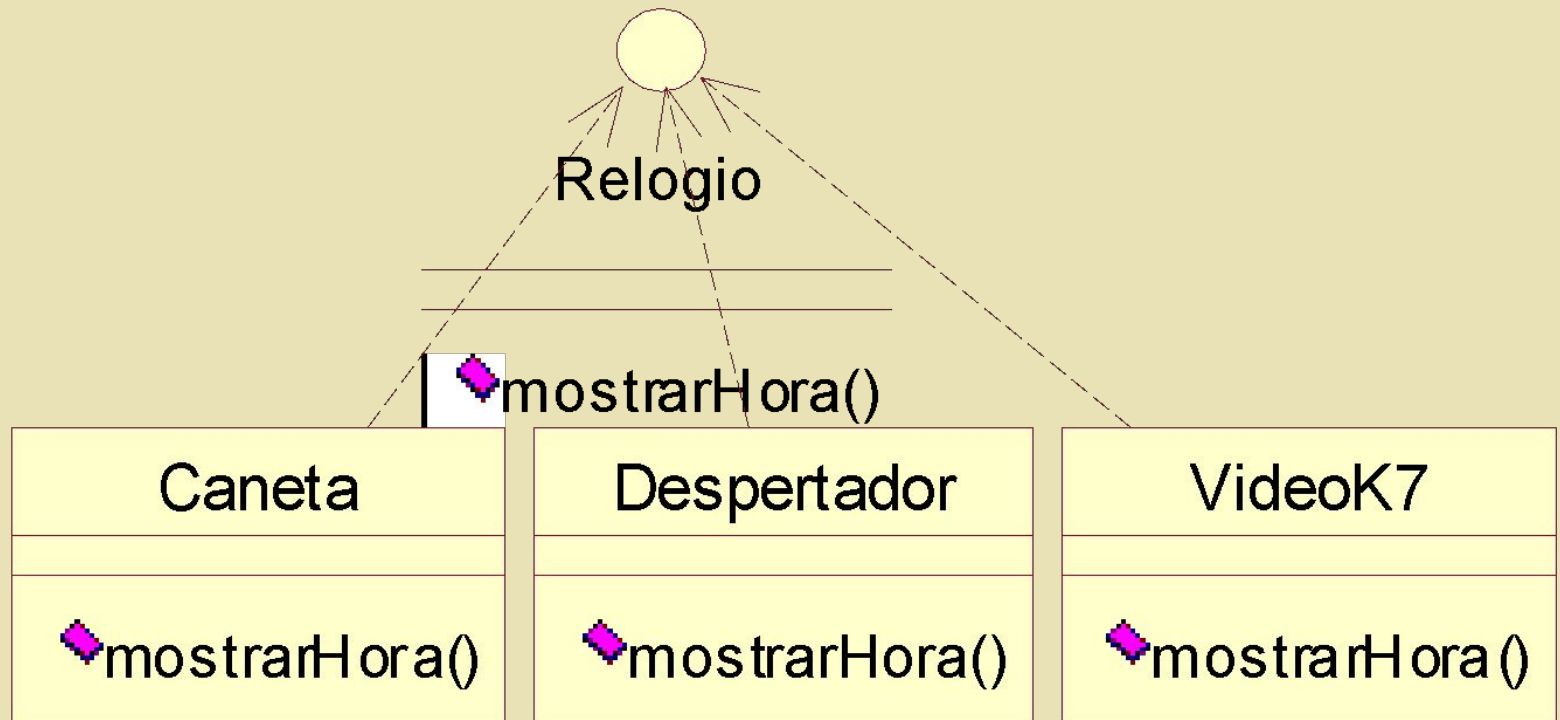


Pessoa
hora(Relógio r)

mostrarHora() não é herdado e sim definido em cada classe que implementa a interface Relógio.

```
public void hora (Relógio r)
{ r.mostrarHora(); }
// seria parte de qualquer objeto com a capacidade de
mostrar a hora.
```

Exemplos de Interface





Criação de Interface

◆ Exemplo:

```
public interface Relogio
{
    public String mostrarHora();
    public String mostrarData();
    public int  mostrarSegundos();
}
```



Implementação de Interface

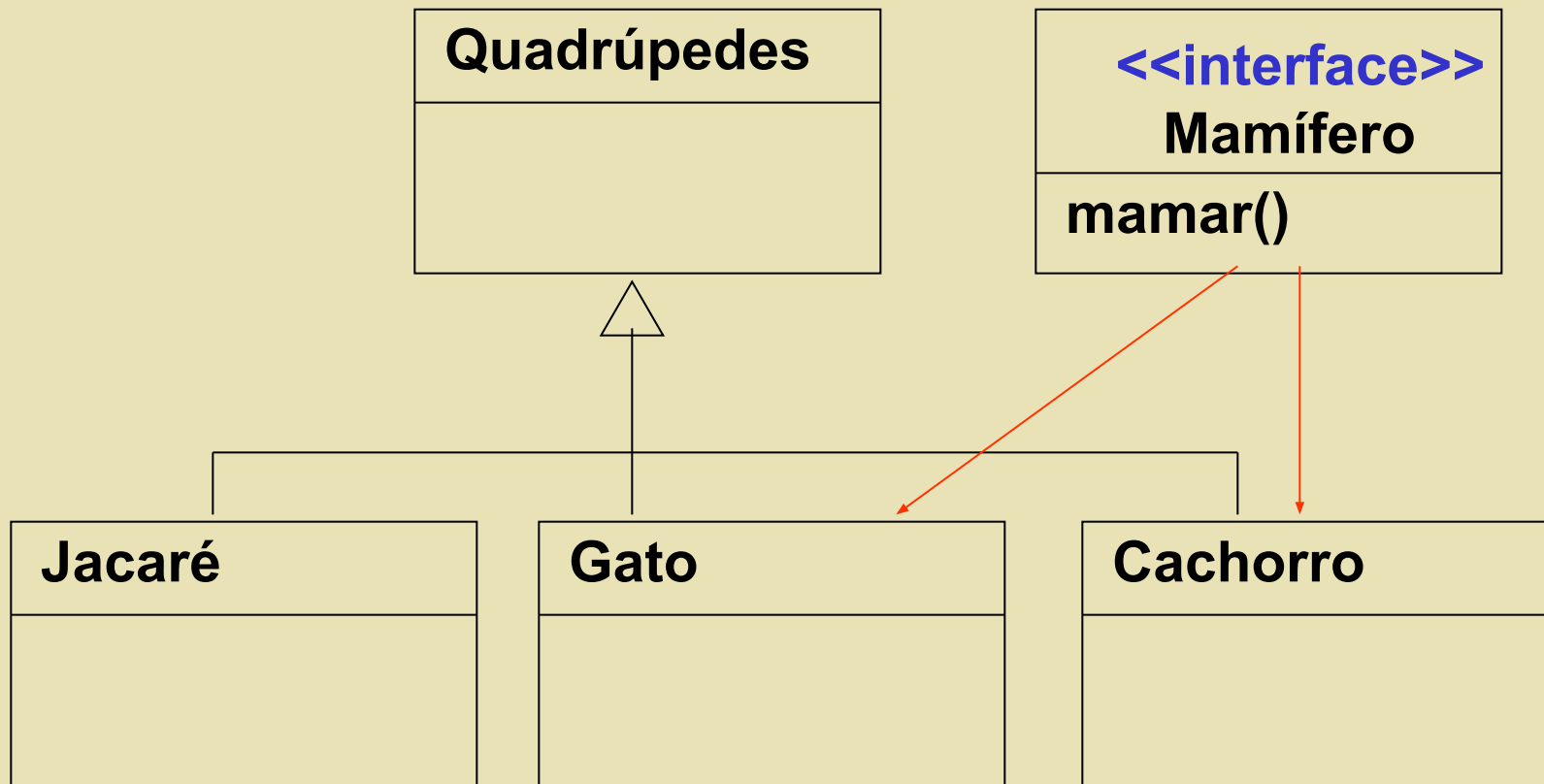
```
public class Caneta implements Relogio
public void escrever(String msg)
{ System.out.println(msg); }

public String mostrarHora()
{ // código p/ apresentação da hora  }

public String mostrarData()
{ // código p/ apresentação da data  }

public int mostrarSegundos()
{ // código p/ apresentação dos
segundos  }
}
```


Exemplo de Interface



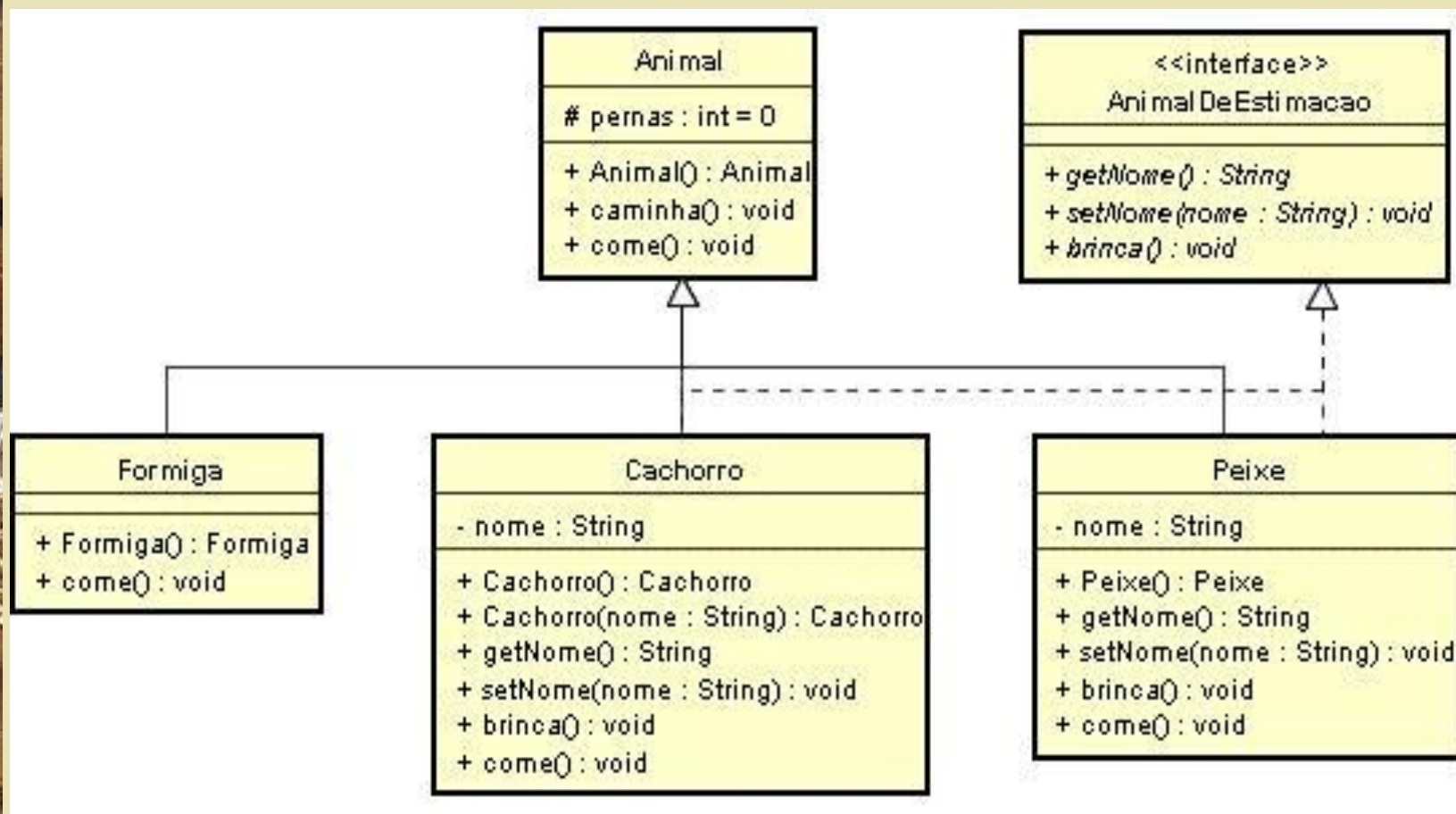


Implementação de Interface

```
public interface Mamifero  
{ public void mamar(); }
```

```
public class Morcego implements Mamifero {  
    public void mamar()  
    {        //.....    }  
}
```

Implementação de Interface



Implementação de Interface

Material de Apoio

Interfaces e Classes Abstratas

<http://slideplayer.com.br/slide/3320266/>

