



# Programação Orientada a Objetos I

**Prof<sup>a</sup>. Angela Abreu Rosa de Sá, Dr<sup>a</sup>.**

---

*Contato: [angelaabreu@gmail.com](mailto:angelaabreu@gmail.com)*

# Material Didático

## Programação Orientada a Objetos

### Sumário

Unidade 1   Fundamentos da orientação a objetos .....	7
Seção 1.1 - Histórico e introdução à orientação a objetos .....	9
Seção 1.2 - Conceitos básicos de orientação a objetos .....	22
Seção 1.3 - Construtores e sobrecarga .....	37
Unidade 2   Estruturas de programação orientadas a objetos .....	59
Seção 2.1 - Estruturas de decisão e controle em Java .....	61
Seção 2.2 - Estruturas de repetição em Java .....	76
Seção 2.3 - Reutilização de classes em Java .....	93
Unidade 3   Exceções, classes abstratas e interfaces .....	111
Seção 3.1 - Definição e tratamento de exceções .....	113
Seção 3.2 - Definição e uso de classes abstratas .....	126
Seção 3.3 - Definição e uso de interfaces .....	141
Unidade 4   Aplicações orientadas a objetos .....	155
Seção 4.1 - Arrays em Java .....	157
Seção 4.2 - Strings em Java .....	173
Seção 4.3 - Coleções e arquivos .....	188

# Conceitos Fundamentais

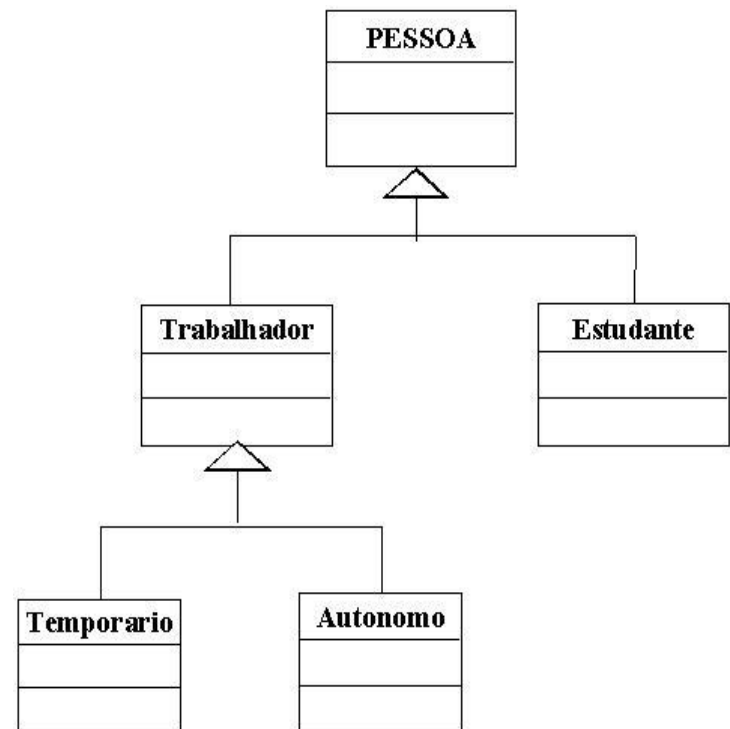
---

- Classe / Objeto
- Construtor
- Atributos
- Métodos
- Sobrecarga
- Encapsulamento
- Herança/Generalização/Especialização
- Polimorfismo

# Herança

- Redução no custo de manutenção;
- Aumento na reutilização de código ;
- Modularização;

*Vantagens*

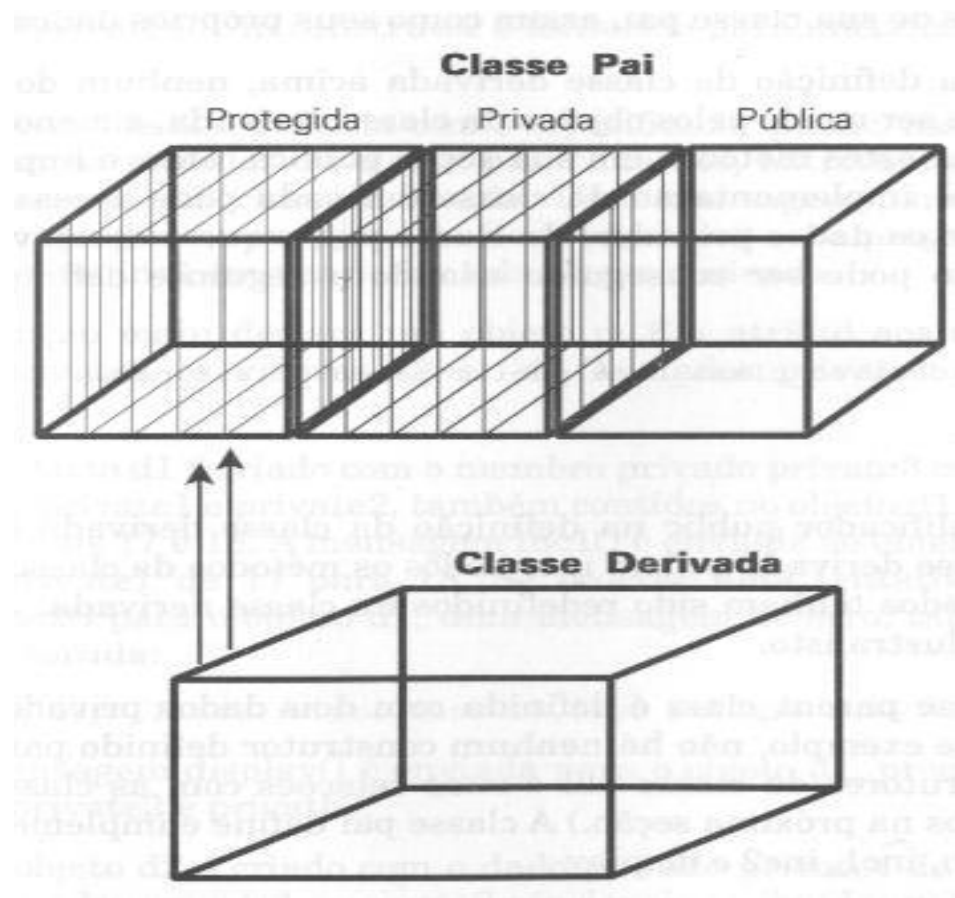


# Herança

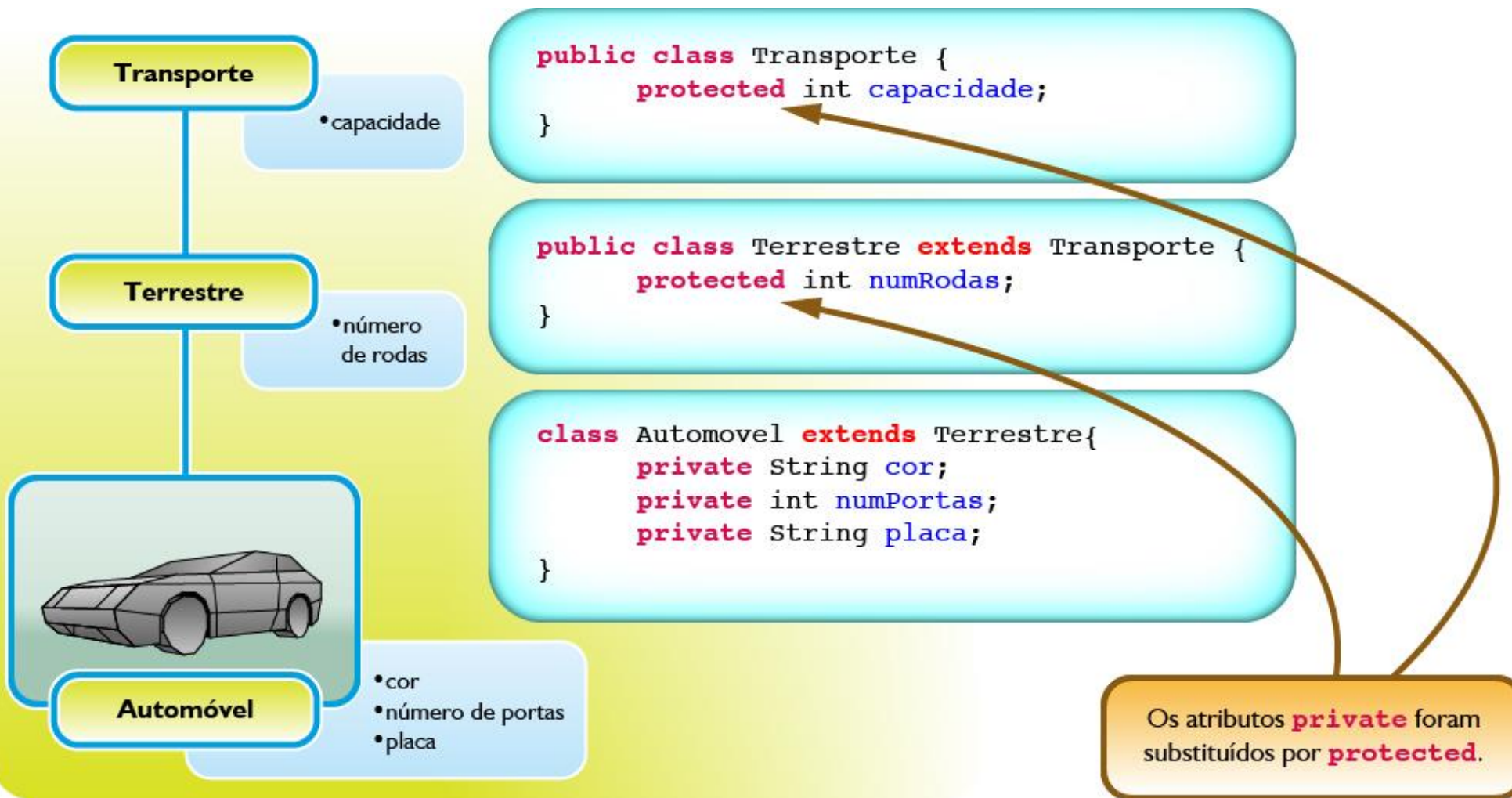
Tipo de acesso	Descrição
<b>Public</b>	Estes atributos e métodos são sempre acessíveis em todos os métodos de todas as classes. Este é o nível menos rígido de encapsulamento, que equivale a não encapsular.
<b>Private</b>	Esses atributos e métodos são acessíveis somente nos métodos da própria classe. Este é o nível mais rígido de encapsulamento.
<b><u>Protected</u></b>	Esses atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses.

# Herança

## Tipos de acesso



# Herança



```
public class Funcionario {  
  
    protected String nome;  
    protected String cpf;  
    protected double salario;  
  
    public Funcionario(String n, String numcpf, double sal)  
    { //inicializar os atributos  
        nome = n;  
        cpf = numcpf;  
        salario = sal;  
    }  
}
```

```
public class Gerente extends Funcionario {  
  
    private int senha;  
    private int numeroDeFuncionariosGerenciados;  
  
    public Gerente (String n, String numcpf, double sal, int s, int numgerenciado)  
    {  
        super(n,numcpf,sal); //chamando o construtor da super classe (classe pai)  
        senha = s;  
        numeroDeFuncionariosGerenciados = numgerenciado;  
    }  
}
```

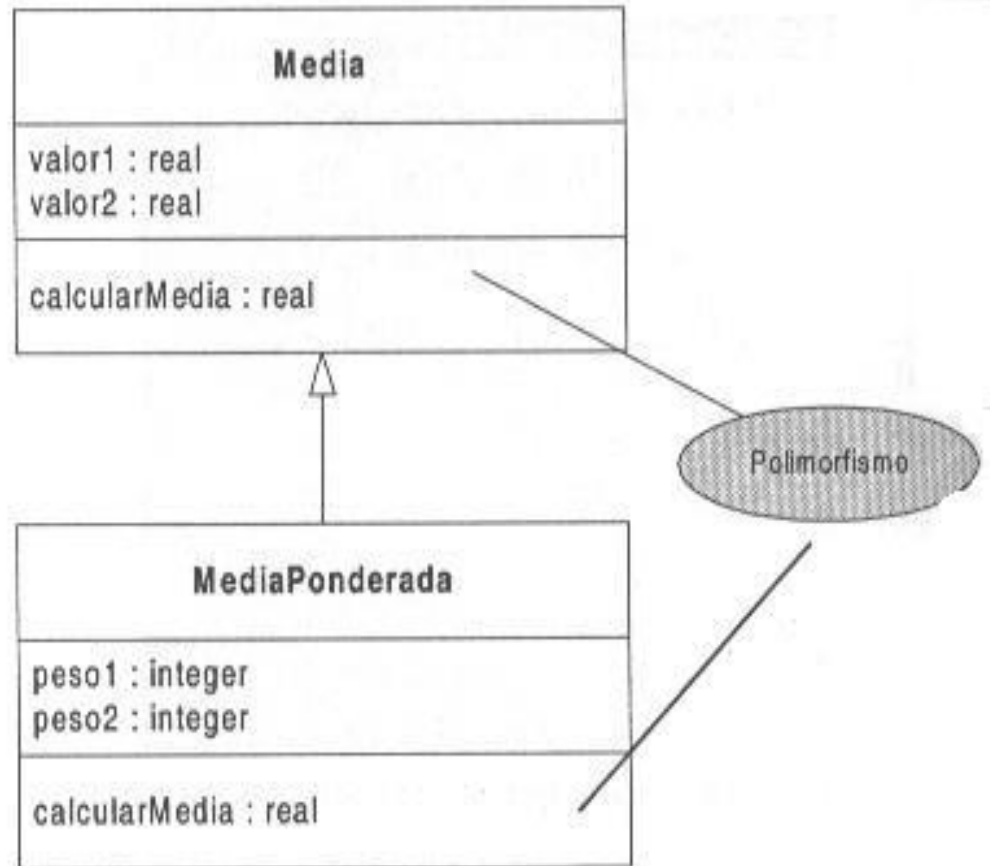


# Polimorfismo

*poli* = muitas,  
*morphos* = formas

**POLIMORFISMO**

**MUITAS FORMAS**



```
public class Funcionario {  
  
    protected String nome;  
    protected String cpf;  
    protected double salario;  
  
    public Funcionario(String n, String numcpf, double sal)  
    { //inicializar os atributos  
        nome = n;  
        cpf = numcpf;  
        salario = sal;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getCpf() {  
        return cpf;  
    }  
  
    public void setCpf(String cpf) {  
        this.cpf = cpf;  
    }  
  
    public double getSalario() {  
        return salario;  
    }  
  
    public void setSalario(double salario) {  
        this.salario = salario;  
    }  
}
```

this

# Orientação a objetos

**Encapsulamento**

**Herança**

**Composição**

**Polimorfismo**

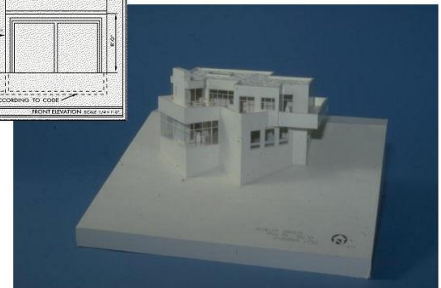
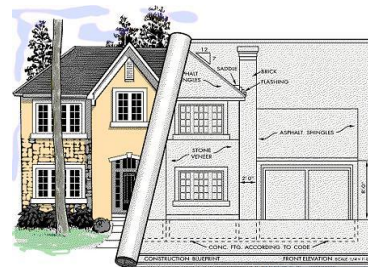
**Princípio da abstração**

# Como iniciar um programa Orientado a Objetos?



## MODELO

Projeto!!!



# Modelagem

A modelagem é uma parte central de todas as atividades que levam à implantação de um “bom” software.

Construímos **modelos** para **compreender melhor o sistema que estamos desenvolvendo**.

Objetivos da Modelagem:

- Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja.
- Os modelos permitem especificar a estrutura (organização) ou o comportamento (dinâmica) de um sistema.
- Os modelos proporcionam um guia para a construção (implementação) do sistema.
- Os modelos documentam as decisões tomadas.

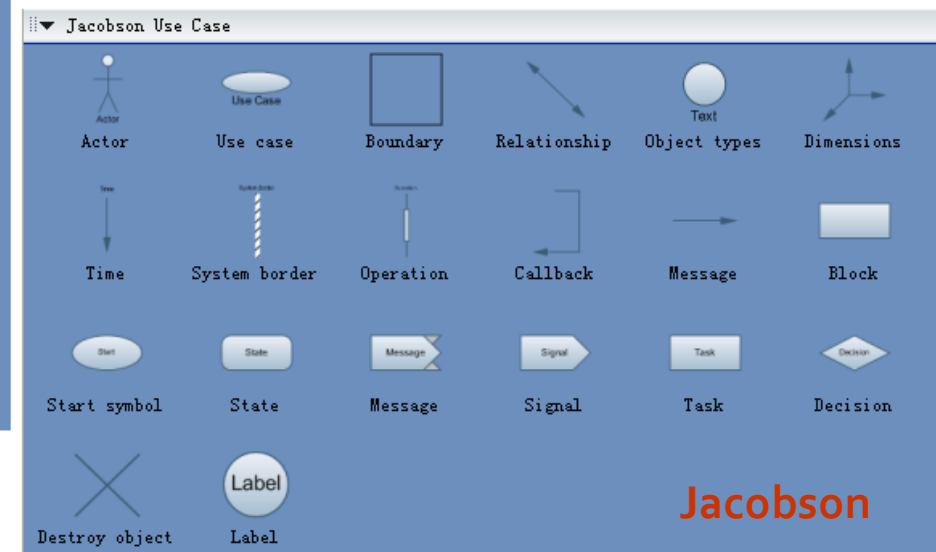
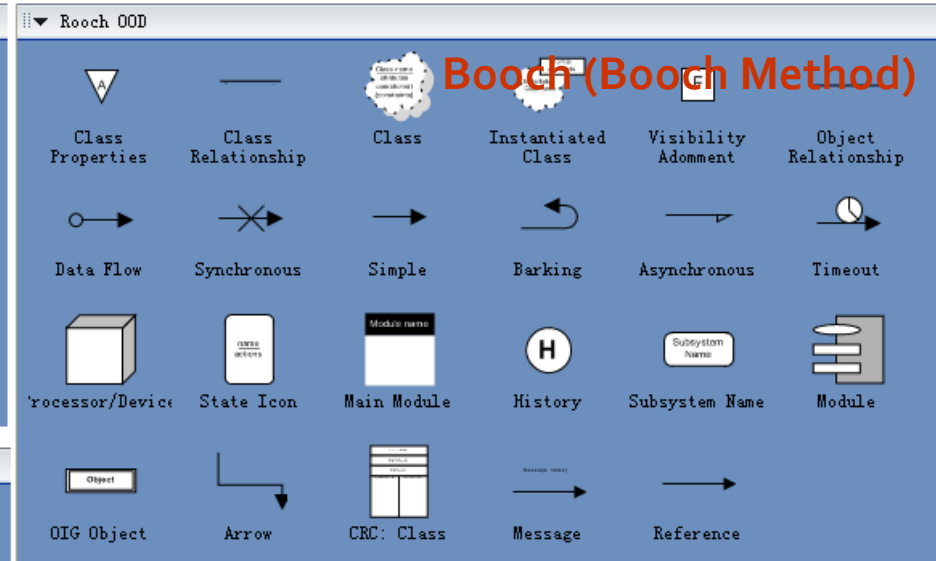
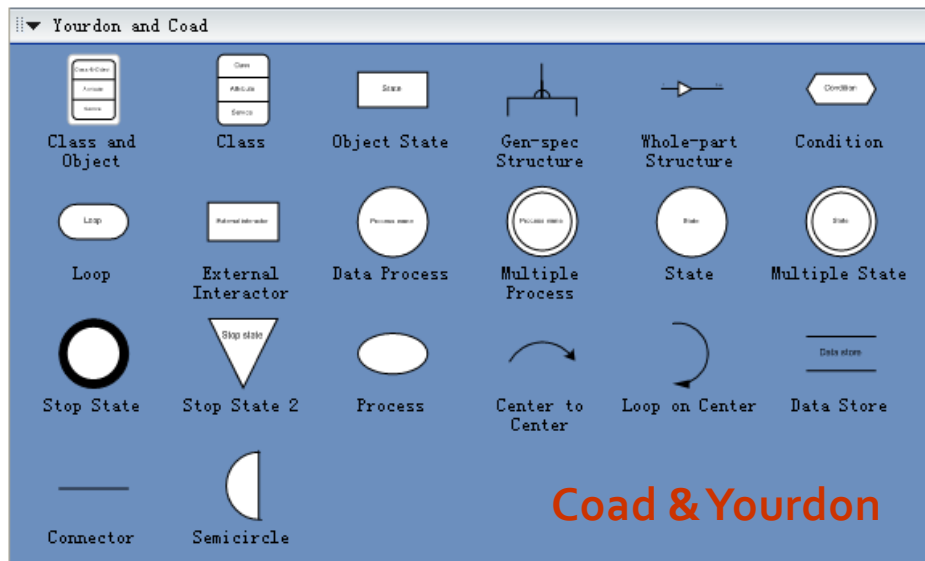
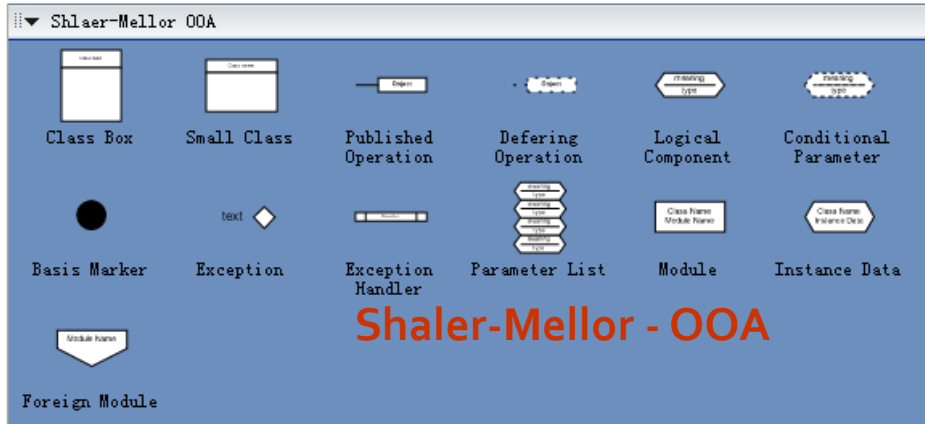
# Modelagem de Sistemas

---

- ◆ 1950/60 – Sistemas ad hoc
  - ◆ Época dos fluxogramas
- ◆ 1970 – Programação estruturada
  - ◆ Tom de Marco, Edward Yourdon
- ◆ 1980 – Análise estruturada moderna
  - ◆ Necessidade de interface mais sofisticada
  - ◆ DFD, DTE, DER
- ◆ 1990 – Análise Orientada a Objetos
  - ◆ Shaler, Mellor, Booch, Jacobson, Rumbaugh

# Modelagem: Histórico

## Técnicas para modelagem



# UML

- Percebeu-se a **necessidade de um padrão** para a modelagem de sistemas, que fosse aceito e utilizado amplamente.
- Surge a **UML (Unified Modeling Language)** em **1996** como a melhor candidata para ser linguagem “unificadora”.
- Desde então, a UML **tem tido grande aceitação pela comunidade** de desenvolvedores de sistemas.





# Histórico – Técnicas de modelagem OO

Ano	Autor (Técnica)
1990	Shaler & Mellor
1991	Coad & Yourdon (OOAD – Object-oriented Analysis and Design)
1993	Grady Booch (Booch Method)
1993	Ivar Jacobson (OOSE – Object Oriented Software Engineering)
1995	James Rumbaugh et al (OMT – Object Modeling Technique)
1996	Wirfs-Brock (Responsability Driven Design)
<b>1996</b>	<b>Surge a UML como a melhor candidata de notações, diagramas e formas de representação</b>

# UML



A **UML** (*Unified Modeling Language*, ou **Linguagem de Modelagem de Objetos Unificada**) é uma linguagem de modelagem padrão para **elaboração da estrutura de projetos de software orientado a objetos**.



É **independente** de linguagens de programação e de processo de desenvolvimento

# UML

- “A UML é a linguagem padrão para **visualizar, especificar, construir e documentar** os artefatos de software de um sistema.”
- Unificação de diversas notações anteriores.)

Aproveitar o **melhor** das características das notações **preexistentes**

# UML

- UML é...
  - uma **linguagem visual**.
  - independente de linguagem de programação.
  - independente de processo de desenvolvimento.
- Um processo de desenvolvimento que utilize a UML como linguagem de modelagem envolve a criação de diversos **diagramas/documentos**.



# UML

## ■ Diagramas:

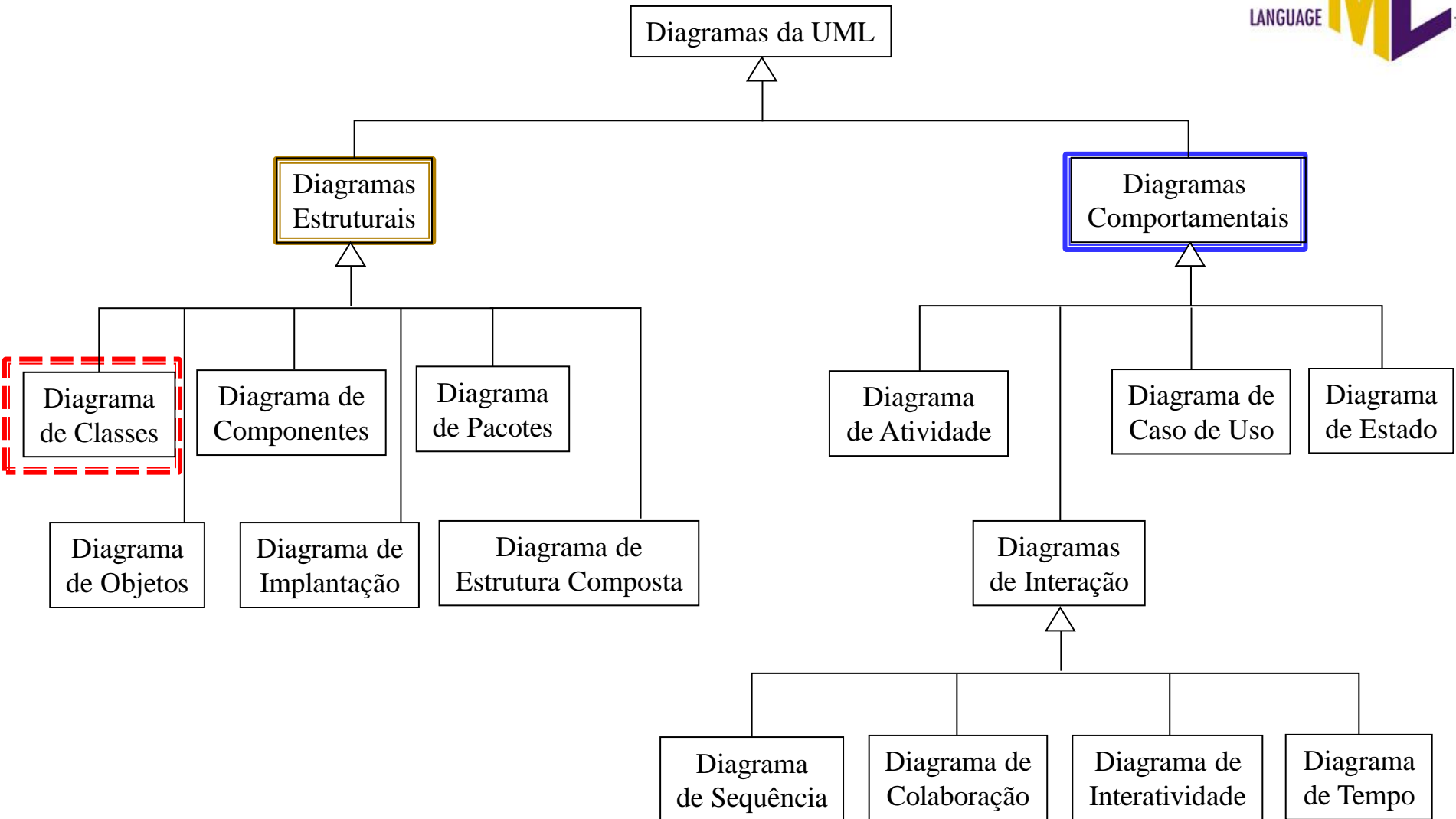
- Os **documentos** gerados em um processo de desenvolvimento são chamados de **artefatos na UML**
- Os artefatos compõe as **visões do sistema**
- A UML define **13 diagramas**
- Esta quantidade de diagramas é justificada pela **necessidade de analisar o sistema por meio de diferentes perspectivas**
- Cada diagrama fornece uma perspectiva parcial do sistema.

# Modelagem de Casos de Uso



- Linguagem de Modelagem que **permite a representação de conceitos do mundo real**, sob a ótica da orientação a objetos.
- A UML permite que desenvolvedores **visualizem** os produtos de seus trabalhos em **diagramas padronizados**

# Diagramas da UML



# Diagrama de Classes

---

- Permite a **visualização das classes** que compõem o sistema: **como as classes estão organizadas?**

## Representa

- **Atributos e métodos** de uma **classe**
- Os relacionamentos entre classes.



# Atributos e Métodos

---

## Atributos

- Devem apresentar o **tipo de dados** a ser armazenado  
Byte, boolean, int, double, char, String, etc.

## Métodos

- São apenas **declarados** neste diagrama  
Diagrama de **Classes não define a implementação**

# Visibilidade

---

- **Pública (+)**

- O atributo ou método pode ser utilizado por qualquer classe

- **Protegida (#)**

- Somente a classe ou sub-classes terão acesso

- **Privada (-)**

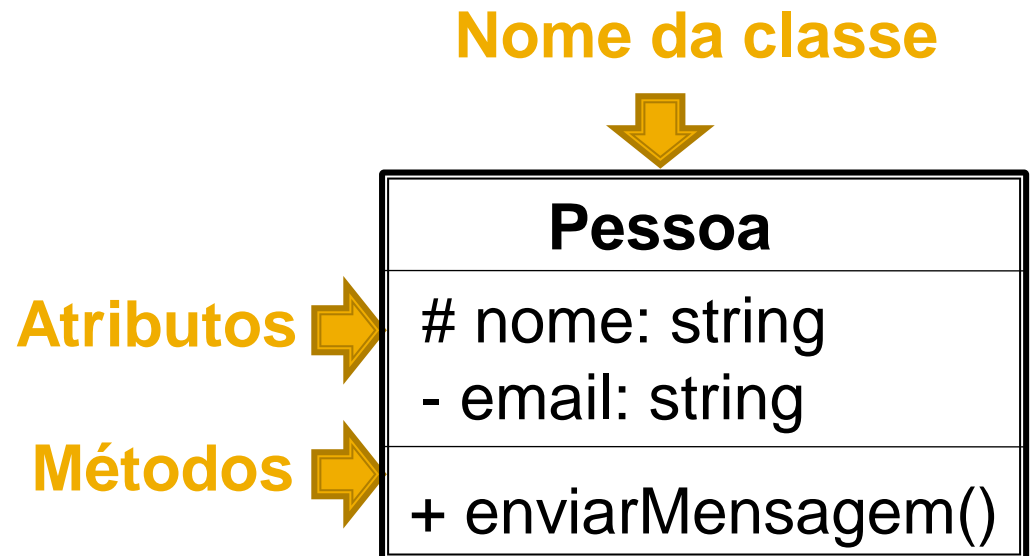
- Somente a classe terá acesso

# Classe

- Uma classe é representada por um retângulo com **três divisões**:

- Nome da Classe
- Atributos da Classe
- Métodos da Classe

Nome da classe
Lista de atributos
Lista de métodos



# Classe

ContaBancaria
numero saldo dataAbertura
criar() bloquear() desbloquear creditar()

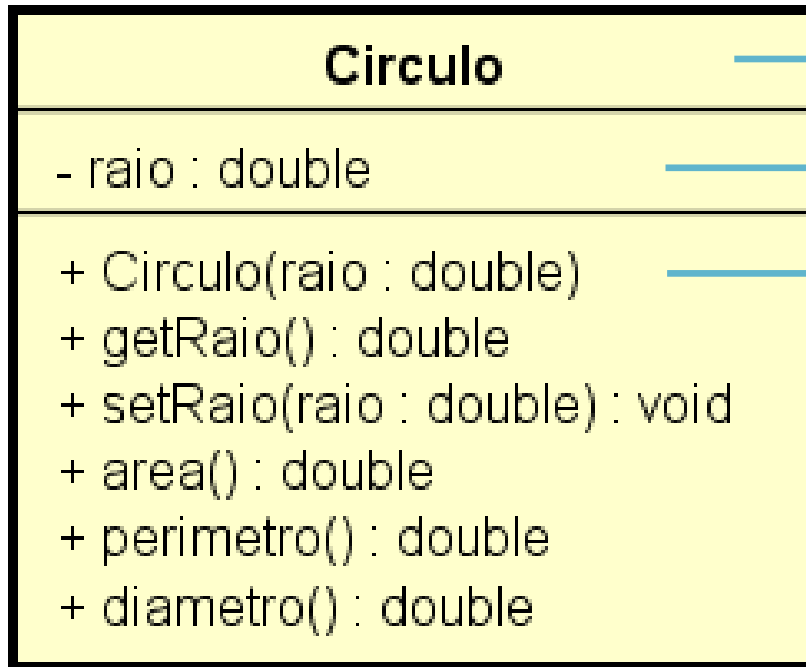
ContaBancaria
-numero:String -saldo:Quantia -dataAbertura: Date
+criar() +bloquear() +desbloquear (in Valor: Quantia) +creditar(in Valor: Quantia)



# Exemplo:

**atributos privado (-):** raio

**métodos públicos (+):** Circulo (construtor), getRaio, setRaio, area, perímetro e diâmetro.



Nome da Classe

Atributo (ou propriedade)

Operações (ou métodos)

# Relacionamento

---

- Classes possuem **relacionamentos** entre elas
  - Compartilham informações
  - Colaboram umas com as outras
- Principais tipos de relacionamentos
  - Associação
  - Agregação / Composição
  - Herança

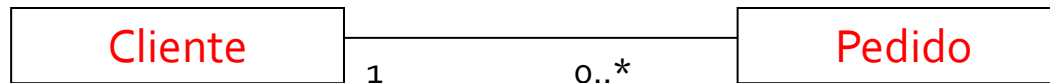
# Associação

## ♦ Multiplicidade:

- ♦ Representa as informações dos **limites inferior e superior da quantidade de objetos** aos quais outro objeto pode estar associado.

Nome	Simbologia
Apenas Um	<b>1</b> (ou <b>1..1</b> )
Zero ou Muitos	<b>0..*</b> (ou <b>*</b> )
Um ou Muitos	<b>1..*</b>
Zero ou Um	<b>0..1</b>

# Exemplo: Associação



- ◆ Pode haver algum objeto da classe Cliente que está associado a *vários objetos* da classe Pedido (representado por \* do 0..\*)
- ◆ Pode haver algum objeto da classe Cliente que *NÃO* está associado a classe Pedido (representado por 0 do 0..\*)
- ◆ Objetos da classe pedido está associado a **UM** e somente um objeto da classe Cliente

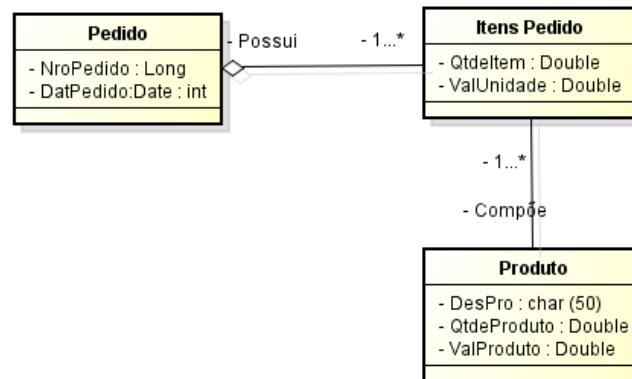
Cliente José tem os pedidos 1, 2 e 3  
Cliente Ana tem os pedidos 4 e 5  
Cliente Maria não tem pedidos



# Agregação

- Demonstra que as informações e um objeto **precisam ser complementadas** por um objeto de outra classe
- Associação Todo-Parte: **objeto-todo** e **objeto-parte**

Um losango na extremidade da classe que contém os *objetos-todo*



# Composição

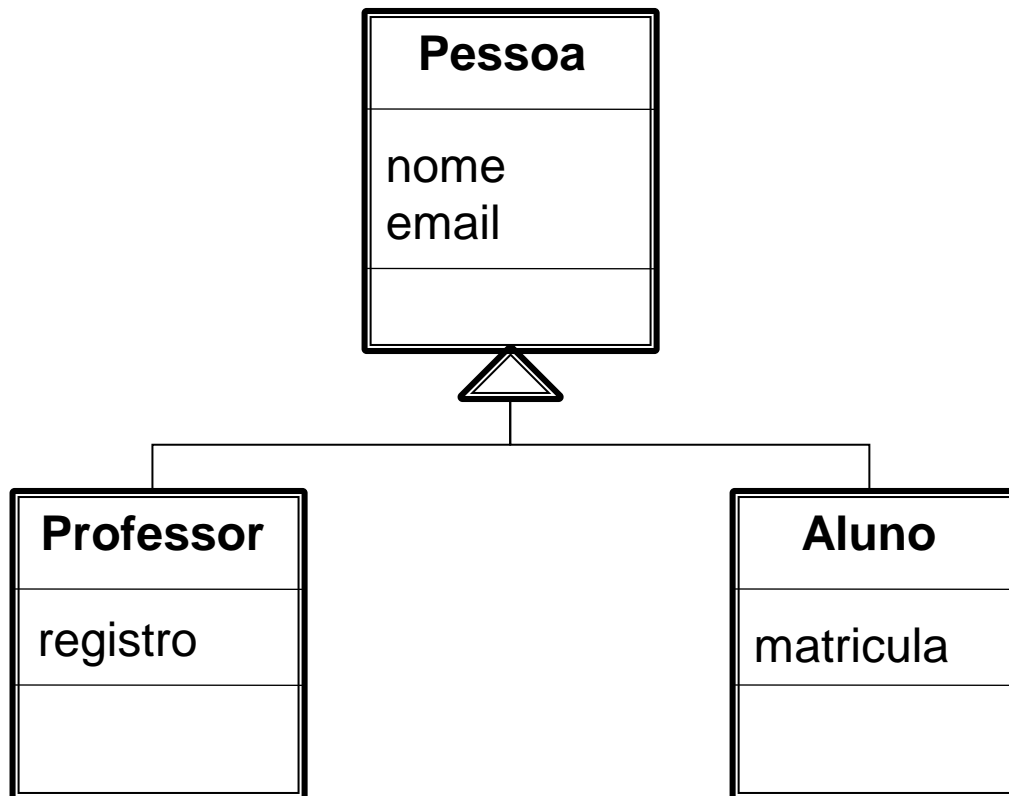
- Uma variação do tipo agregação
- Representa um **vínculo mais forte** entre objetos-todo e objetos-parte
- Objetos-parte **têm** que pertencer ao objeto-todo

O todo **não existe (ou não faz sentido)** sem a parte

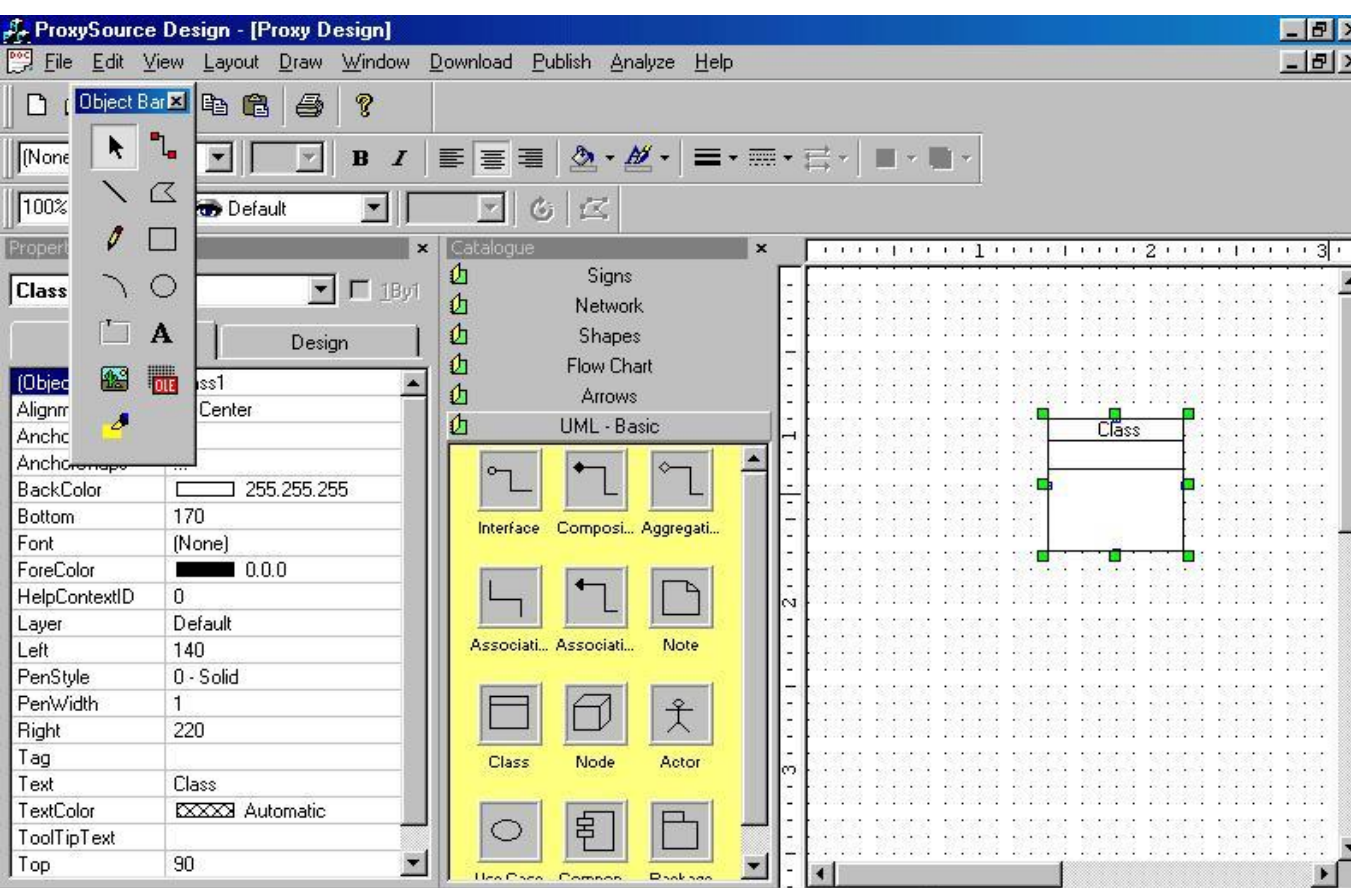


# Herança

- Identificar classes-mãe (gerais) e classes-filhas (especializadas)
- Atributos e métodos definidos na classe-mãe são **herdados** pelas classes-filhas

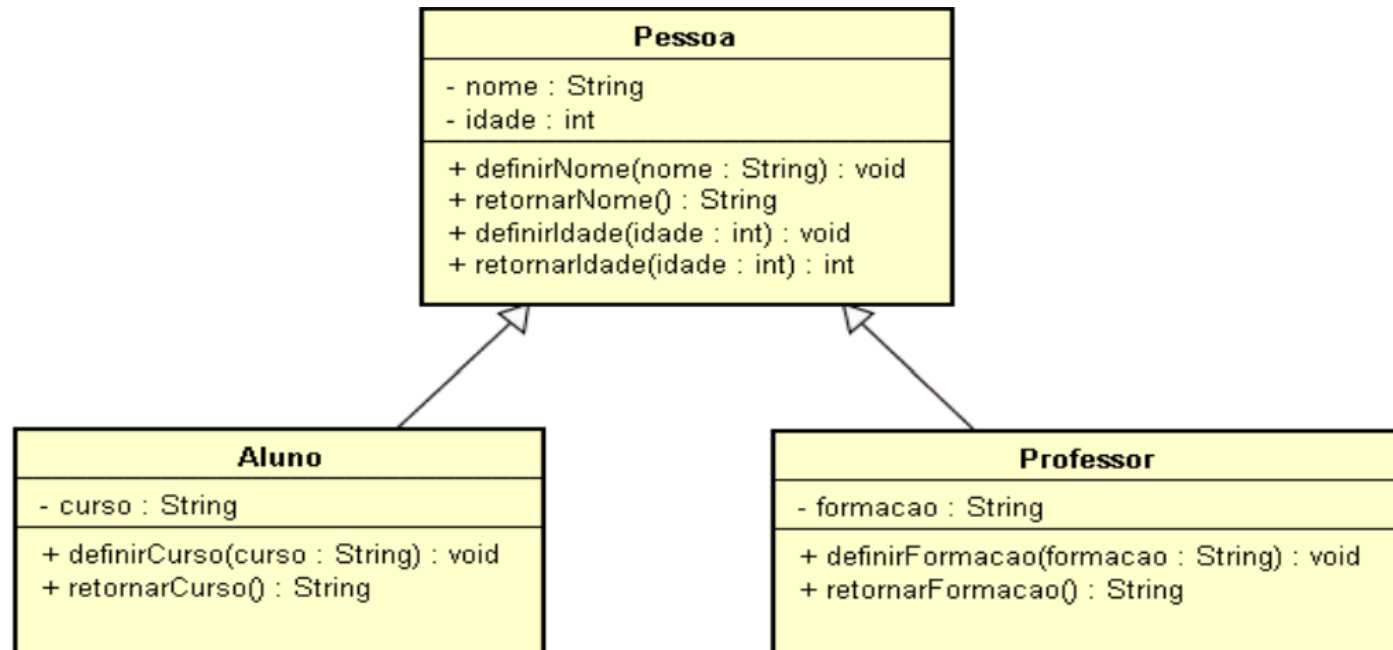


# Modelagem UML

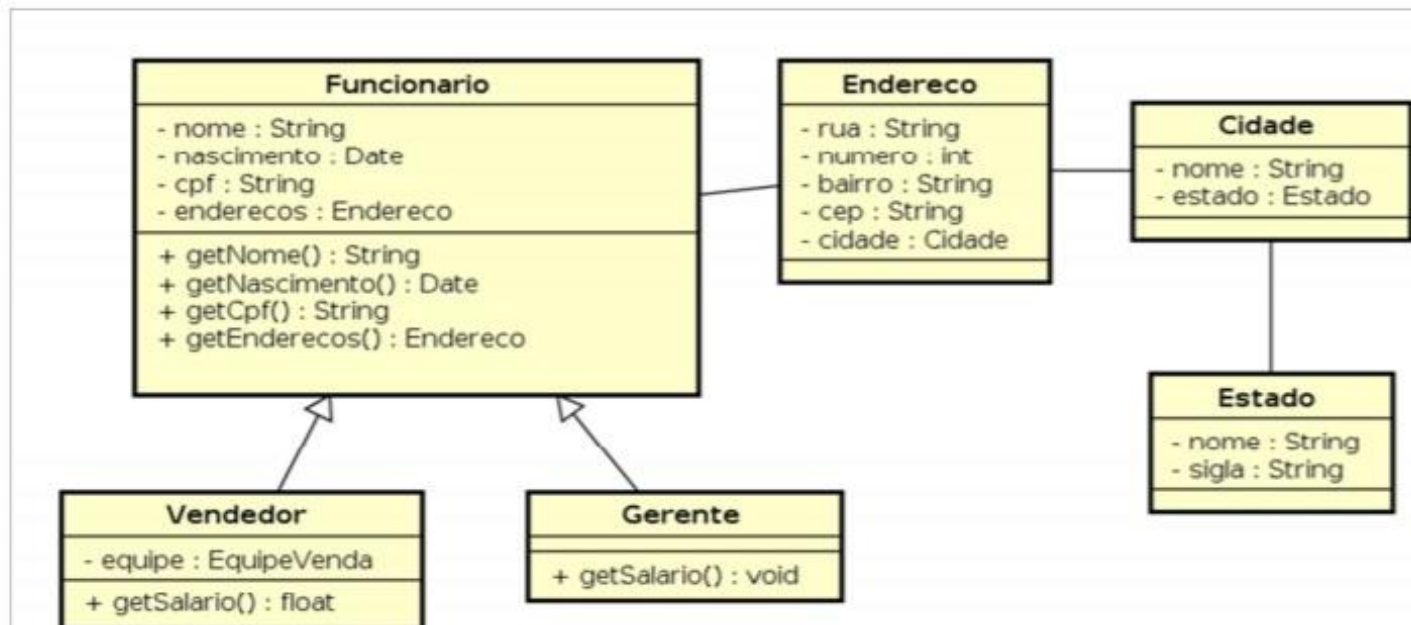


- Star UML
- Drawio
- Proxy design
- Argo UML
- Visio
- Visual UML
- Poseidon UML
- Top Coder UML
- Violet UML
- jUML
- Visual Paradigm
- etc

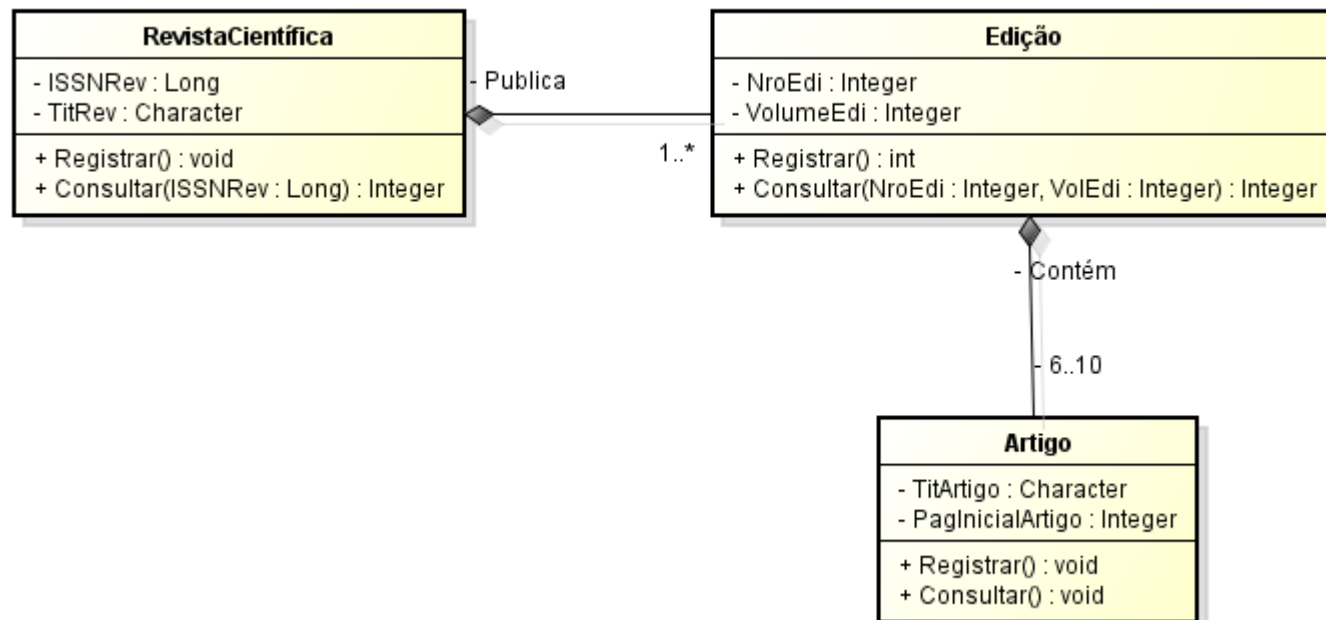
# Exemplo



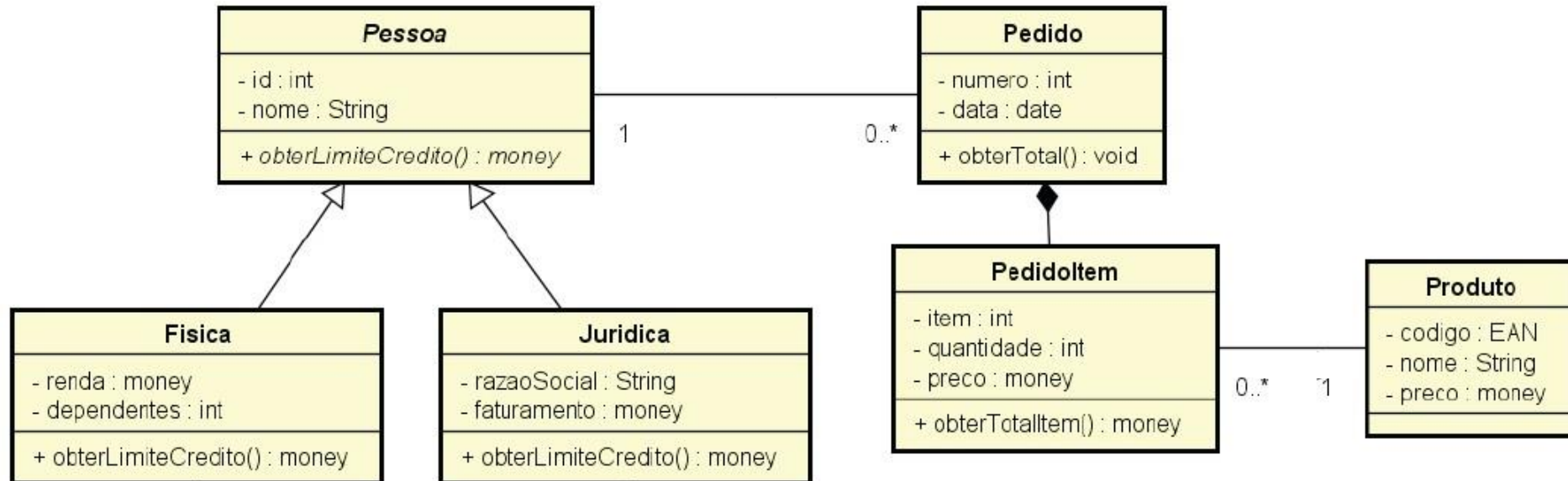
# Exemplo



# Exemplo

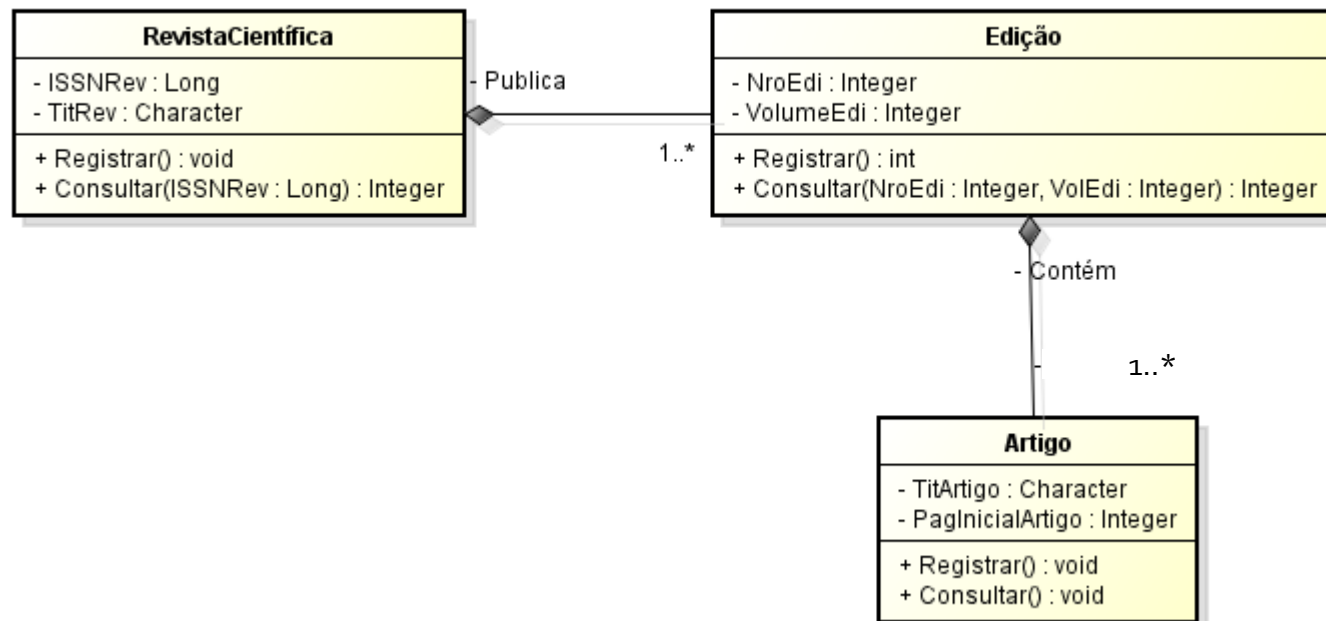


# Exemplo





# Exemplo



# Material Didático

## Programação Orientada a Objetos

### Sumário

Unidade 1   Fundamentos da orientação a objetos .....	7
Seção 1.1 - Histórico e introdução à orientação a objetos .....	9
Seção 1.2 - Conceitos básicos de orientação a objetos .....	22
Seção 1.3 - Construtores e sobrecarga .....	37
Unidade 2   Estruturas de programação orientadas a objetos .....	59
Seção 2.1 - Estruturas de decisão e controle em Java .....	61
Seção 2.2 - Estruturas de repetição em Java .....	76
Seção 2.3 - Reutilização de classes em Java .....	93
Unidade 3   Exceções, classes abstratas e interfaces .....	111
Seção 3.1 - Definição e tratamento de exceções .....	113
Seção 3.2 - Definição e uso de classes abstratas .....	126
Seção 3.3 - Definição e uso de interfaces .....	141
Unidade 4   Aplicações orientadas a objetos .....	155
Seção 4.1 - Arrays em Java .....	157
Seção 4.2 - Strings em Java .....	173
Seção 4.3 - Coleções e arquivos .....	188

# Array ou Vetor

```
<tipo> <nome_do_vetor>[tamanho];
```

```
int idade;
```

```
int [ ]idade = new int[10]
```

35	69	10	25	37	52	36	15	41	23
----	----	----	----	----	----	----	----	----	----

Índice/posição

0 1 2 3 4 5 6 7 8 9

```
idade [0] = 35;
```

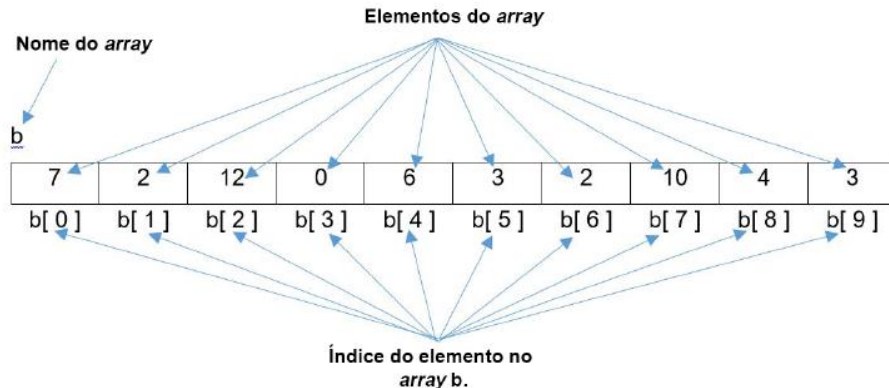
```
Idade [8] = 41;
```

```
int x;
```

```
x = idade[2] + idade[9];
```

# Array ou Vetor

Figura 4.1 | Array de 10 elementos




```
int [ ]idade = new int[10]
```

- O **tamanho** de um *array* **não pode ser modificado** após sua criação.
- Alterações no tamanho devem ser feitas **com a criação de outro *array***, de tamanho diferente, para que os elementos do *array* original possam ser copiados nele.
- **Inserções e deleções** em um *array* **não são operações simples.**


```
public class Principal01 {  
    public static void main(String[] args) {  
  
        Scanner scan = new Scanner(System.in);  
        Aluno []vetorObjetosAluno; //VETOR DE OBJETOS ALUNO  
  
        System.out.print("Digite a quantidade de alunos: ");  
        int qtdeAlunos = scan.nextInt();  
  
        vetorObjetosAluno = new Aluno[qtdeAlunos]; //CRIAR O VETOR DE OBJETOS  
  
        String nome;  
        int nota1, nota2, nota3;  
  
        for (int i = 0; i < qtdeAlunos; i++)  
        {  
            System.out.println("***** Dados do Aluno ***** ");  
            System.out.print("Digite o nome do aluno: ");  
            nome = scan.next();  
            System.out.print("Digite a nota 1: ");  
            nota1 = scan.nextInt();  
            System.out.print("Digite a nota 2: ");  
            nota2 = scan.nextInt();  
            System.out.print("Digite a nota 3: ");  
            nota3 = scan.nextInt();  
  
            //INSTANCIAR O OBJETO PARA CADA POSIÇÃO DO VETOR  
            vetorObjetosAluno[i] = new Aluno(nome, nota1,nota2,nota3);  
        }  
  
        System.out.println("***** Boletim final ***** ");  
        for (int i = 0; i < qtdeAlunos; i++)  
        {  
            System.out.println("Aluno " + vetorObjetosAluno[i].RetornaNome() + " está " + vetorObjetosAluno[i].ResultadoFinal());  
        }  
    }  
}
```

```
private Jogador []VetorJogadores;  
private int totaljogadores ;
```




```
public TimeVolei()
```

```
{  
    VetorJogadores = new Jogador[6];  
    totaljogadores = 0;  
}
```



```
public void InsereJogador(String n, int i, float d)
```

```
{  
    if(totaljogadores < 6)  
    {  
        VetorJogadores[totaljogadores] = new Jogador(n,i,d);  
        totaljogadores++;  
    }  
    return;  
}
```



```
public String NomeJogadorMenorAltura()
```

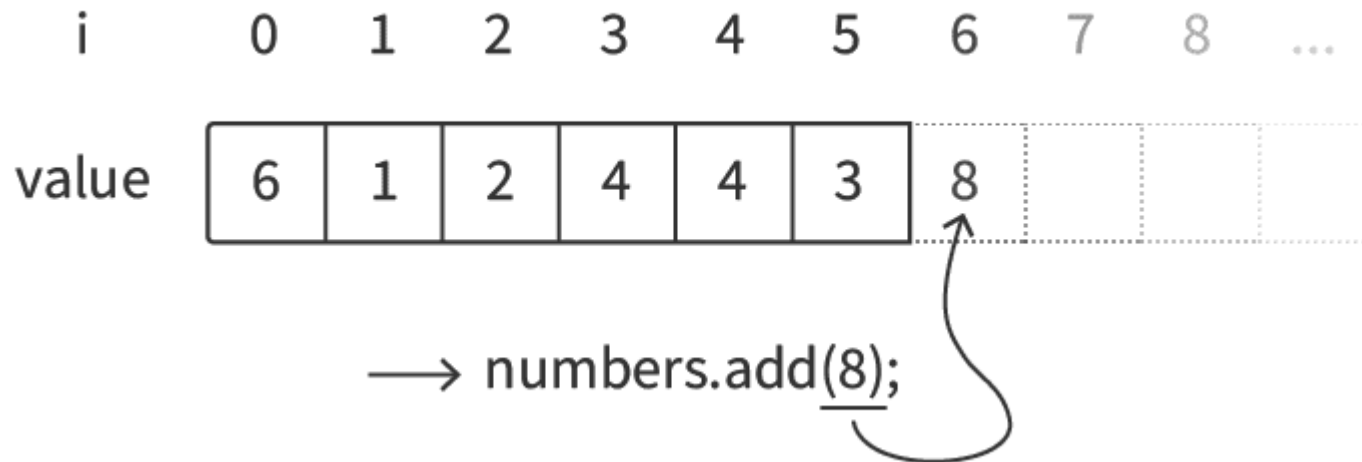
```
{  
    float MenorAltura = 999999;  
    String NomeJogador = "";  
  
    for (int i = 0; i < totaljogadores; i++)  
    {  
        if(VetorJogadores[i].RetornaAltura() < MenorAltura)  
        {  
            MenorAltura = VetorJogadores[i].RetornaAltura();  
            NomeJogador = VetorJogadores[i].RetornaNome();  
        }  
    }  
    return NomeJogador;  
}
```

# ArrayList

```
import java.util.ArrayList;
```

São Arrays| Vetores que crescem **dinamicamente**: não precisamos especificar o tamanho!

```
ArrayList<INTEGER> numbers = new ArrayList<INTEGER>();
```



```
public static void main(String[] args) {
```

```
    Scanner entrada = new Scanner(System.in);
```

```
    ArrayList<Integer> lista = new ArrayList<Integer>(); //Cria um arraylist de INTEIROS
```

```
    int numero;
```

```
    int indice;
```

```
    while (true)
```

```
    { //insere mais um elemento na lista até que -1 seja digitado
```

```
        System.out.print("Insira um número inteiro positivo para ser incluído ou -1 para terminar: ");
```

```
        numero = entrada.nextInt();
```

```
        if (numero==-1)
```

```
        { break;
```

```
        }
```

```
        lista.add(numero); //adicionar um número na lista
```

```
    }
```

```
    System.out.println("\nA lista que você criou contém os seguintes elementos: ");
```

```
    for (int i: lista)
```

```
    {
```

```
        System.out.print(i + " ");
```

```
    }
```

```
    System.out.print("\nInforme o índice do elemento que você deseja retirar da lista: ");
```

```
    indice = entrada.nextInt();
```

```
    lista.remove(indice); //remover o o numero que está na posicao "indice"
```

```
    System.out.println("Agora sua lista ficou assim: ");
```

```
    for (int i: lista)
```

```
    {
```

```
        System.out.print(i + " ");
```

```
    }
```

```
}
```

```
<terminated> principal3 [Java Application] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (19 de out. de 2022 13:46)
Insira um número inteiro positivo para ser incluído ou -1 para terminar: 5
Insira um número inteiro positivo para ser incluído ou -1 para terminar: 6
Insira um número inteiro positivo para ser incluído ou -1 para terminar: 9
Insira um número inteiro positivo para ser incluído ou -1 para terminar: 8
Insira um número inteiro positivo para ser incluído ou -1 para terminar: -1
```

```
A lista que você criou contém os seguintes elementos:
```

```
5 6 9 8
```

```
Informe o índice do elemento que você deseja retirar da lista: 2
```

```
Agora sua lista ficou assim:
```

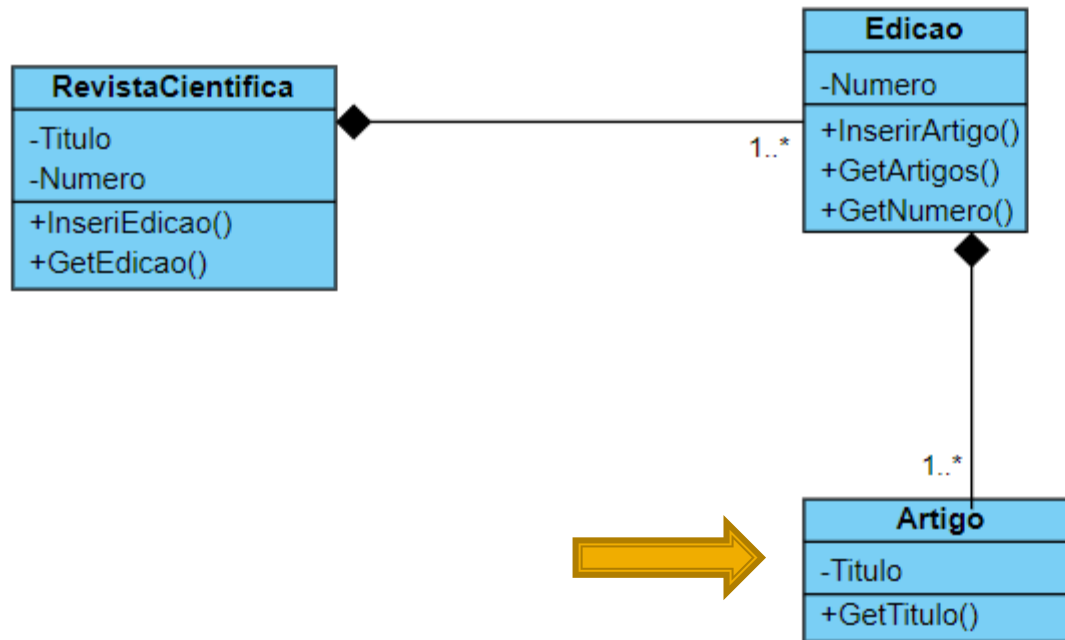
```
5 6 8
```






# Exercícios 😊

# Exercício 1



## Java Class

 Type already exists.Source folder:  Package:  ☐ Enclosing type:  Name: 

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:  Interfaces:  

Which method stubs would you like to create?

- ☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

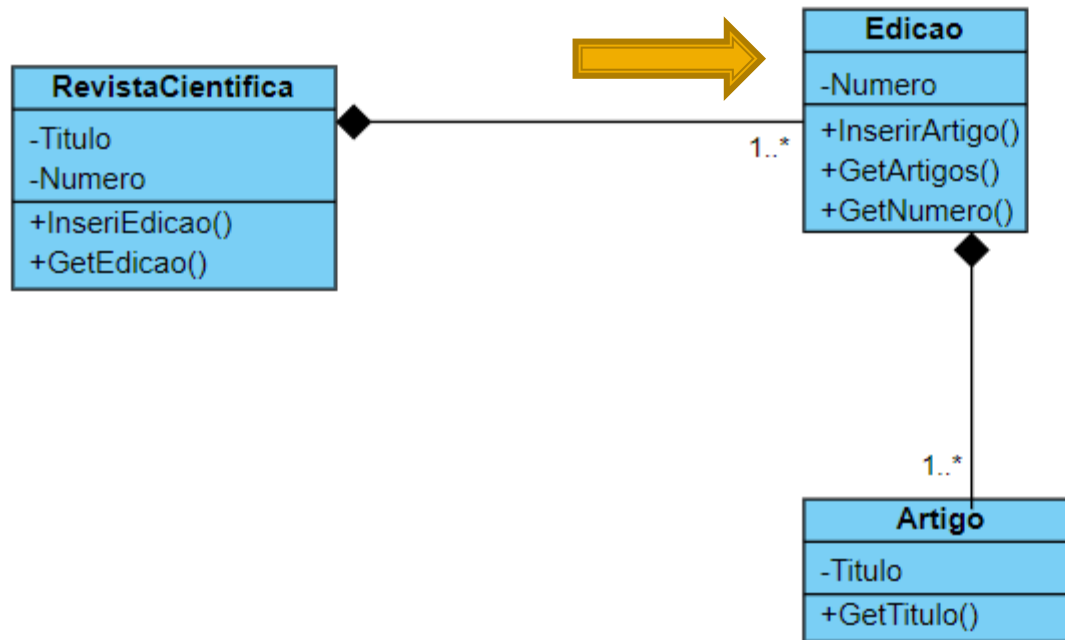
Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments



```
public class Artigo {  
  
    private String Titulo;  
  
    public Artigo(String Titulo)  
    {  
        this.Titulo = Titulo;  
  
    }  
  
    public String getTitulo() {  
        return Titulo;  
    }  
  
}
```

# Exercício 1



## Java Class

⚠ The use of the default package is discouraged.



Source folder: Exercicio2/src

Browse...

Package:

(default)

Browse...

☐ Enclosing type: principal3

Browse...

Name:

Edicao

Modifiers:

☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

java.lang.Object

Browse...

Interfaces:

Add...

Remove

Which method stubs would you like to create?

- ☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments

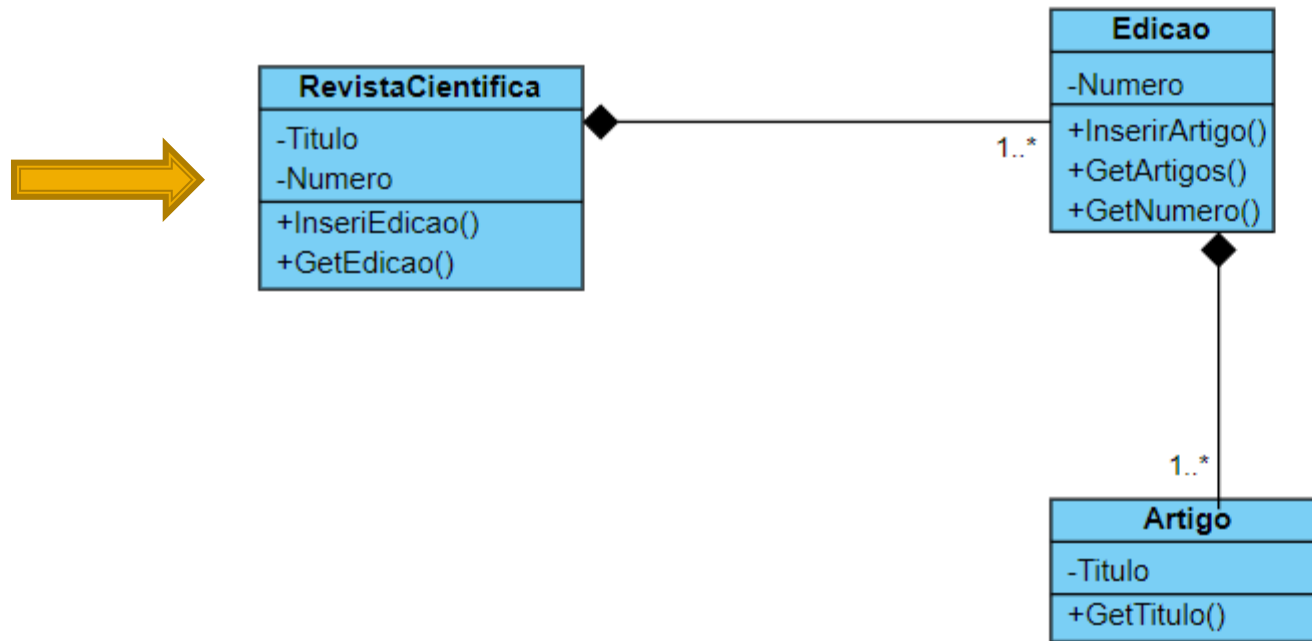


Finish

Cancel

```
public class Edicao {  
  
    private int Numero;  
  
    private ArrayList<Artigo> Array_Artigo = new ArrayList<Artigo>(); //Cria um arraylist de objetos Artigo  
  
    public Edicao(int Numero)  
    {  
        this.Numero = Numero;  
    }  
  
    public void InsereArtigo(String Titulo)  
    {  
        Artigo objArtigo = new Artigo( Titulo);  
        Array_Artigo.add(objArtigo);  
    }  
  
    public ArrayList<Artigo> getArray_Artigo() {  
        return Array_Artigo;  
    }  
  
    public int getNumero() {  
        return Numero;  
    }  
}
```

# Exercício 1





## Java Class

⚠ The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

Which method stubs would you like to create?

- ☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments



```
import java.util.ArrayList;

public class RevistaCientifica {


    private int Numero;
    private String Titulo;
    private ArrayList<Edicao> Array_Edicao = new ArrayList<Edicao>(); //Cria um arraylist de objetos Edicao

    public RevistaCientifica(int Numero, String Titulo)
    {
        this.Numero = Numero;
        this.Titulo = Titulo;
    }

    public void InserirEdicao(Edicao objEdicao )
    {
        Array_Edicao.add(objEdicao);
    }

    public ArrayList<Edicao> getArray_Edicao() {
        return Array_Edicao;
    }
}
```

## Java Class

 The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

Which method stubs would you like to create?

- ☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- ☐ Generate comments



```
import java.util.ArrayList;
import java.util.Scanner;

public class Principal {

    public static void main(String[] args) {

        //criando o objeto da revista "Ciência e Tecnologia"
        RevistaCientifica objetoRevista = new RevistaCientifica(563, "Ciência e Tecnologia");

        int numero = 0;
        String titulo;
        Scanner leituraTeclado = new Scanner(System.in);

        while(numero != -1)
        {
            System.out.print("*** Cadastro de Edicao ***: \n");
            System.out.print("Digite o Numero da Edicao:");
            numero = leituraTeclado.nextInt();

            if(leituraTeclado.hasNextLine()) //limpar o conteúdo que possa ter ficado no teclado
                leituraTeclado.nextLine();

            if(numero == -1) //encerrar o cadastro de edições?
                break;

            //criar o objeto edição
            Edicao objetoEdicao = new Edicao(numero);

            //em cada edição, temos 5 artigos artigos
            System.out.print("\n*** Cadastro dos artigos ***: \n");
            for(int i = 0; i < 5; i++)
            {
                System.out.print("\n Digite o titulo do artigo " + (i+1) + ":");
                titulo = leituraTeclado.nextLine();



                //inserir o artigo na Edicao
                objetoEdicao.InsereArtigo(titulo);
            }
            //inserir a Edição na revista
            objetoRevista.InserirEdicao(objetoEdicao);
        }
    }
}
```

```

System.out.print("\n ### CONTEUDO DA REVISTA #####");
//mostrar todas as edições da revista e todos os artigo de cada edição
for (Edicao edicao: objetoRevista.getArray_Edicao())
{
    // mostrar o número da edição
    System.out.print("\n ----- Edição: " + edicao.getNumero() );

    //mostrar todos os artigo da edição
    for (Artigo artigo: edicao.getArray_Artigo())
    {
        // mostrar o título do artigo
        System.out.print("\n ** Artigo: " + artigo.getTitulo());
    }
}

```

```

*** Cadastro de Edicao ****:
Digite o Numero da Edicao:1

*** Cadastro dos artigos ****:

Digite o titulo do artigo 1:Titulo 1
Digite o titulo do artigo 2:Titulo 2
Digite o titulo do artigo 3:Titulo 3
Digite o titulo do artigo 4:Titulo 4
Digite o titulo do artigo 5:Titulo 5
*** Cadastro de Edicao ****:
Digite o Numero da Edicao:2

*** Cadastro dos artigos ****:

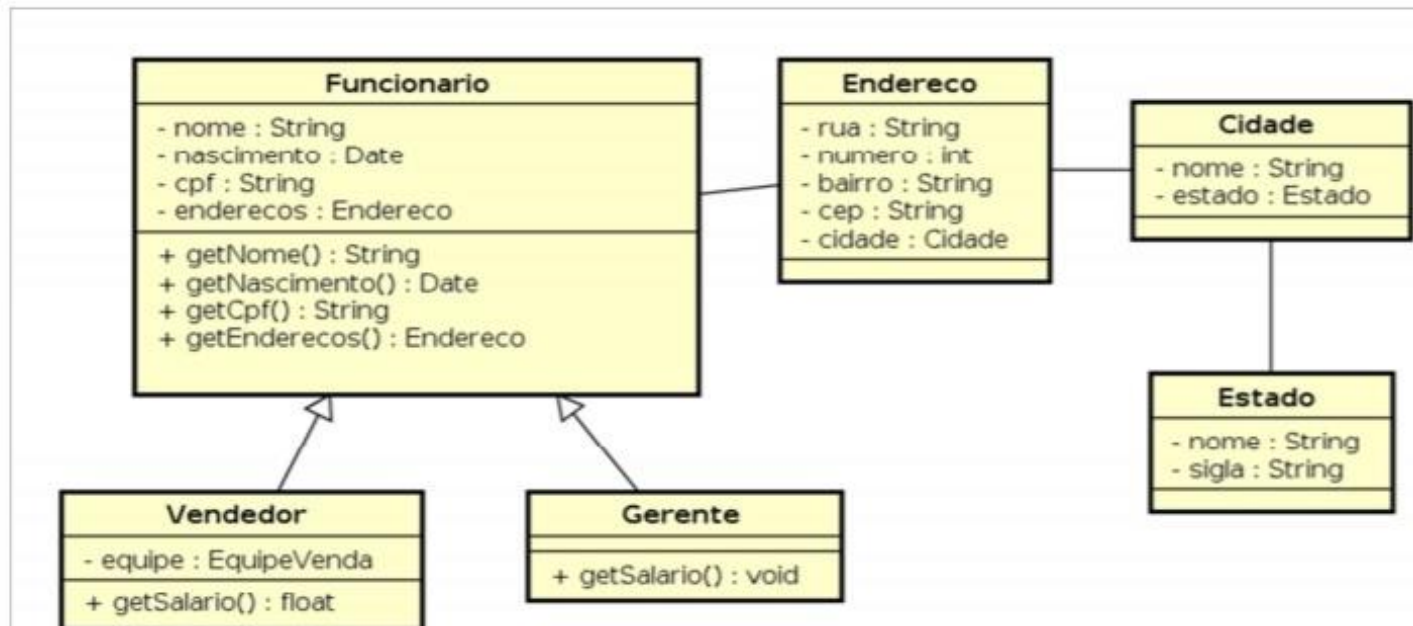
Digite o titulo do artigo 1:artigo 1
Digite o titulo do artigo 2:artigo 2
Digite o titulo do artigo 3:artigo 3
Digite o titulo do artigo 4:artigo 4
Digite o titulo do artigo 5:artigo 5
*** Cadastro de Edicao ****:
Digite o Numero da Edicao:-1

### CONTEUDO DA REVISTA #####
----- EdiçDo: 1
** Artigo: Titulo 1
** Artigo: Titulo 2
** Artigo: Titulo 3
** Artigo: Titulo 4
** Artigo: Titulo 5
----- EdiçDo: 2
** Artigo: artigo 1
** Artigo: artigo 2
** Artigo: artigo 3
** Artigo: artigo 4
** Artigo: artigo 5

```

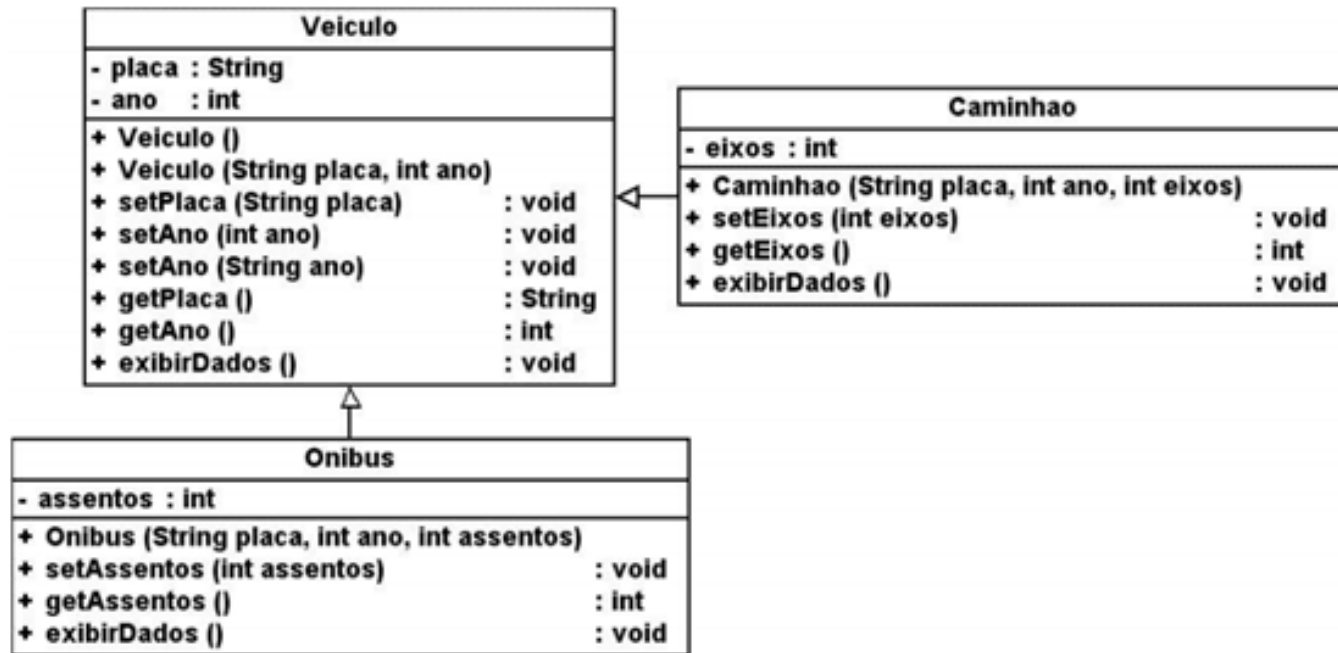
# Exercício 2

Implemente um programa para fazer um cadastro completo dos vendedores e do gerente de uma Empresa de Equipamentos Tecnológicos - **utilizar ARRAYLIST**. Ao final do cadastro, o programa deverá apresentar um relatório completo de todas as informações dos funcionários.



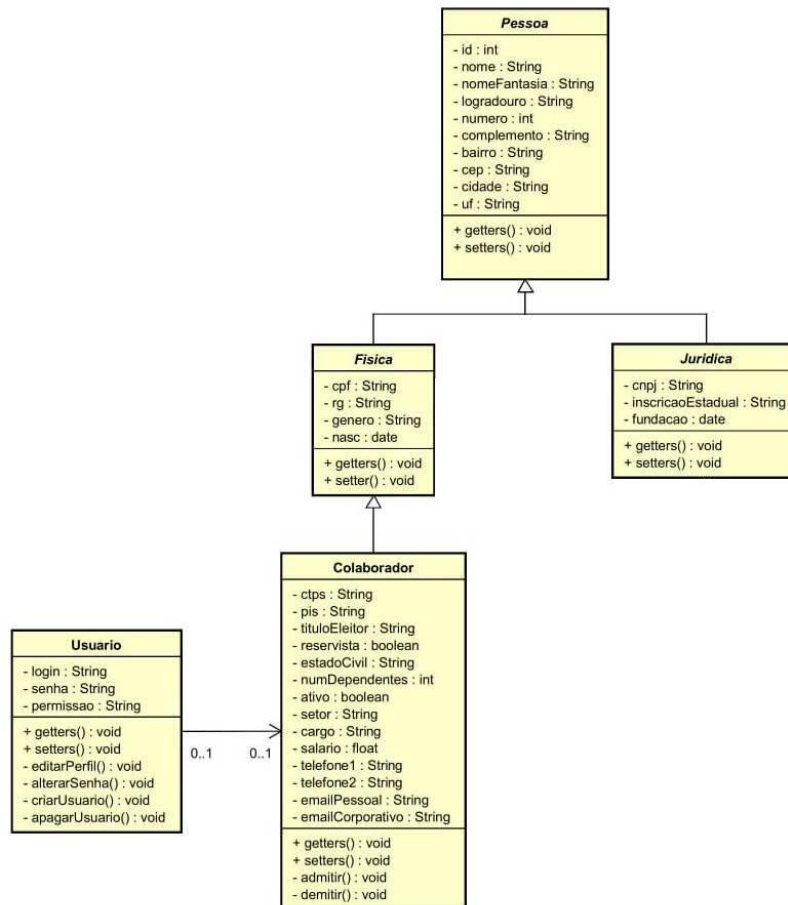
# Exercício 3

Implemente um programa para cadastrar veículos (caminhão ou ônibus). O usuário deverá cadastrar quantos veículos que ele desejar – **utilizar ARRAYLIST**. Após o cadastro, o programa deverá exibir a informação de todos os veículos cadastrados



# Exercício 4

Implemente um programa para cadastrar as pessoas (pessoa física, pessoa jurídica ou colaborador) que trabalham na Faculdade de Computação **utilizar ARRAYLIST**. Ao final do cadastro, o programa deverá apresentar um relatório completo de todas as informações dos funcionários.







# Muito Obrigada!

**Profª. Angela Abreu Rosa de Sá, Drª.**

---

*Contato: [angelaabreu@gmail.com](mailto:angelaabreu@gmail.com)*