



Universidade
de Fortaleza

Programação Orientada a Objeto

Prof. Pedro Pinheiro
Universidade de Fortaleza – UNIFOR

Lógica de Programação

O que é um computador?

- ✓ É uma máquina que, a partir de uma entrada, realiza um número muito grande de cálculos matemáticos e lógicos, gerando uma saída.
- ✓ Os primeiros "computadores" eram humanos que calculavam tabelas de logaritmos ou trajetórias para canhões, seguindo procedimentos bem definidos.

Hardware e dispositivos

- ✓ A linguagem nativa do computador é codificada numericamente, de forma binária:
 - **Bit** → Pode assumir valores 0 ou 1.
 - **Byte** → Agrupamento de 8 bits em uma palavra.
 - Letras e símbolos são representados por números.

Introdução à Lógica de Programação

- ✓ Lógica é a técnica de encadear pensamentos para atingir determinado objetivo;
Sequência Lógica: são passos executados até atingir um objetivo ou solução de um problema.
- ✓ Instruções é um conjunto de regras ou normas definidas para a realização ou emprego de algo e que indica a um computador uma ação elementar a executar.
- ✓ Sequência de passos, precisos e bem definidos, para a realização de uma tarefa. Não pode ser redundante, nem subjetivos e não ambíguo;
- ✓ Algoritmos podem ser especificados de várias formas, inclusive em português.

Exercícios:

1. Crie uma sequência lógica para tomar banho.
2. Faça um algoritmo para somar dois números e multiplicar o resultado pelo primeiro número.
3. Descreva com detalhes a sequência lógica para trocar um pneu de um carro.
4. Faça um algoritmo para trocar uma lâmpada.





Algoritmos- Pseudocódigo

- ✓ São independentes das linguagens de programação;
- ✓ Devem ser fáceis de interpretar e de codificar;
- ✓ Devem ser o intermediário entre a linguagem falada e a linguagem de programação.
- ✓ Regras para construção:

- Usar somente um verbo por frase;
- Imaginar que está desenvolvendo para pessoas que não trabalham com informática;
- Usar frases curtas e simples;
- Ser objetivo;

Algoritmos- Diagrama de Blocos ou Fluxograma

- ✓ É uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento;
- ✓ Pode-se definir uma sequência de símbolos, com significado bem definido;
- ✓ Principal função é facilitar a visualização dos passos de um processamento.

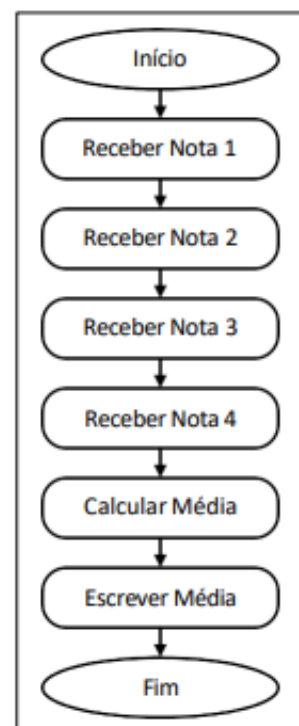
Símbolo	Função
 TERMINAL	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
 PROCESSAMENTO	Processamento em geral Exemplo: Cálculo de dois números
 ENTRADA DE DADO MANUAL	Indica entrada de dados através do Teclado Exemplo: Digite a nota da prova 1
 EXIBIR	Mostra informações ou resultados Exemplo: Mostre o resultado do cálculo

Exemplo: Calcular a média aritmética dos alunos. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

- ✓ Quais são os dados de entrada? Qual será o processamento a ser utilizado? Quais serão os dados de saída?

Algoritmo:

1. Receba a nota da prova 1
2. Receba a nota da prova 2
3. Receba a nota da prova 3
4. Receba a nota da prova 4
5. Some todas as notas e divida o resultado por 4
6. Mostre o resultado da divisão



Exercícios: Construa um algoritmo para pagamento de comissão de vendedores de peças, levando-se em consideração que sua comissão será de 5

Primeiro programa em C

- ✓ Um programa em C é um arquivo texto, contendo declarações e operações da linguagem. Isto é chamado de código fonte.

```

1 #include <stdio.h>
2
3 int main() {
4     printf("Hello, world!\n");
5 }

```

- ✓ Para executar um programa a partir do seu código fonte é necessário compilá-lo, gerando código binário ou executável. Este pode ser executado como qualquer outro programa de aplicação.

O que são erros de compilação?

- ✓ Caso o programa não esteja de acordo com as regras da linguagem, erros de compilação ocorrerão. Ler e entender estes erros é muito importante.

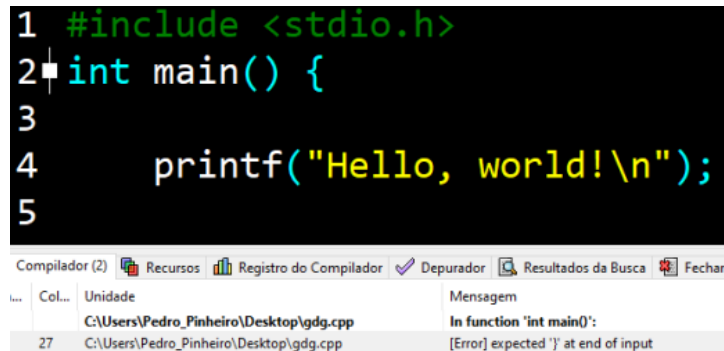


Figura 1: Erros de compilação

- ✓ Acontecem quando o comportamento do programa diverge do esperado e podem acontecer mesmo quando o programa compila corretamente.

Um exemplo mais complexo

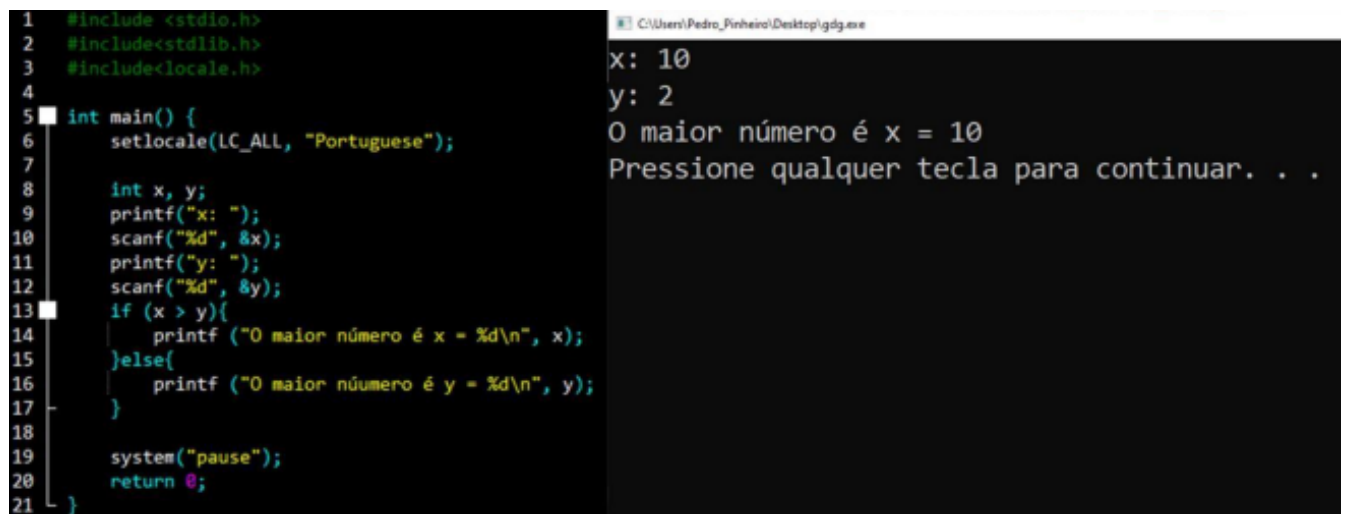


Figura 2: Exemplo mais complexo

Declarar, Ler e Escrever uma Variável:

```
1 /* Exemplo 1: */
2 int x1, y1, result1;           // Declarando as Variaveis globais
    do tipo Int
3 int x = 1, y = 1, result = 0; // Declaracao de Variaveis globais
    atribuindo Valores iniciais
4 float result2;                // Variavel Global do tipo float
5
6 // ENTRADA OU LEITURA DE DADOS
7 scanf("%d", &x1);              // Ler Variavel x1
8 scanf("%d", &y1);              // Ler Variavel y1
9 scanf("%d %d", &x1, &y1);      // ler Variavel x1 e y1
10
11 // SAIDA OU ESCRITA DE DADOS
12 printf("Calculadora de soma: \n"); // Escrever (\n ->
    Pular linha)
13 printf("O valor da soma e: %d ", result1); // Escrever
14 printf("O valor da soma e: %f ", result2); // Escrever
15 printf("O valor da soma e: %.2f ", result2); // Escrever
```

Listing 1: Declarar, Ler e Escrever uma Variável

Estruturas de Decisão

```
1 /* ESTRUTURAS DE DECISAO "if" ou "if else" */
2 if(/*Condicao 1*/){
3     // Blocos de Comandos // se
4 }
5 else if(/*Condicao 2*/){
6     // Blocos de Comandos // senao se
7 }
8 else{
9     // Blocos de Comandos // senao
10 }
11
12 // Quando o "if e else" possui apenas uma instrucao pode ser
    escrito sem {}
13 if(/*Condicao 1*/)
14     // Um Comando
15 else
16     // Um Comando
17
18 // Switch case
19
20 // char operador; // Caractere %c
21 int Operador;
22 printf("Qual operador deseja escolher?\n");
23 scanf("%d", &Operador); // scanf("%c", &operador);
24
25 switch(Operador){
```

```

26     case 1:  // '+'
27         // Bloco de Comandos
28         break;
29     case 2:  // '-'
30         // Bloco de Comandos
31         break;
32     case 3:  // '*'
33         // Bloco de Comandos
34         break;
35     default:
36         // Bloco de Comandos // Caso nenhuma das
37         opcoes seja escolhida

```

Listing 2: Estruturas de Decisão

Estruturas de Repetição

```

1  /* ESTRUTURAS DE REPETICAO "WHILE" */
2  while(/*Condicao*/){ // Enquanto a condicao for verdadeira o
3      while executa
4      // Bloco de Comandos
5  }
6  do{ // Executa oCodigo pelo menos uma vez
7      // Bloco de Comandos
8  }while(/*Condicao*/);
9
10
11 /* ESTRUTURAS DE REPETICAO "FOR" */
12 // Obs: Ex: "i--" -> i = i - 1 (Decrementa) ou "i++" -> i = i + 1 (
13 // Incrementa)
14 for (i = 0; i > 0; i++ /*i--*/){ // for (var = inicio; condicao;
15     incremento)
16     // Bloco de Comandos // Caso necessario
17     break;

```

Listing 3: Estruturas de repeticao

Vetores e Matrizes

```

1  // CRIAR VETOR
2  /* Um vetor e uma sequencia de varios valores do mesmo tipo,
3     armazenados
4     sequencialmente na memoria, e fazendo uso de um mesmo nome de
5     variavel
6     para acessar esses valores. Um vetor tambem pode ser entendido
7     logicamente

```

```

5     como uma lista de elementos de um mesmo tipo. */
6
7 tipo variavel[indice];
8
9 int indice = 10;
10 int vetor[indice];
11
12 scanf("%d", &vetor[indice] );
13 printf("%d", vetor[indice]);
14
15 int indice;
16 int vetor[100];
17 ...
18 for (indice = 0; indice < 100; indice++) {
19     // Bloco de comandos
20 }
21
22 printf("%d", vetor[indice]);
23 scanf("%d", &valores[indice] );

```

Listing 4: Vetores e Matrizes

Introdução a Java

Situação Problema: Quais eram os seus maiores problemas quando programava na década de 1990? ponteiros? gerenciamento de memória? organização? falta de bibliotecas? ter de reescrever parte do código ao mudar de sistema operacional? custo financeiro de usar a tecnologia?

Histórico da linguagem Java:

- ✓ Uma das grandes motivações para a criação da plataforma Java era de que essa linguagem fosse usada em pequenos dispositivos, como: (tvs, videocassetes, aspiradores, liquidificadores e outros)
- ✓ Lançamento focado no uso em clientes web (browsers) para rodar pequenas aplicações (applets).
- ✓ Hoje em dia o Java ganhou destaque no lado do servidor.
- ✓ A página principal do Java é: www.oracle.com/technetwork/java
- ✓ A Sun criou um time (conhecido como Green Team) para desenvolver inovações tecnológicas em 1992.

Características da linguagem Java:

- ✓ Especificação bem definida de uma Linguagem de POO, desenvolvida pela Sun Microsystems.

- ✓ Ambientes de execução de software, presentes em SO, browsers, celulares e eletrodomésticos.
- ✓ Uma coleção de classes, componentes e frameworks (APIs) para o desenvolvimento de aplicações multiplataforma.
- ✓ O conjunto de ferramentas que permite o desenvolvimento de aplicativos Java (JDK – Java Development Kit).
- ✓ A interação com os usuários é feita pela Máquina Virtual Java (JVM);
- ✓ A linguagem Java é executada em diferentes aplicações como, por exemplo:
 - Desktop: (Eclipse, IntelliJ);
 - Web: JSP (Java Server Pages);
 - Embarcadas: Software de receptores de TV a cabo;
 - Dispositivos moveis: Usam o Java ME (Micro Edition);
 - Android: Java é a principal linguagem para desenvolvimento;
 - Applets: aplicativos embutidos no navegador, como: teclados virtuais.
- ✓ Possui várias tecnologias:
 - Java SE (Standard Edition): Mais usada pela comunidade;
 - Java EE (Enterprise Edition): Usada principalmente para aplicações empresariais e internet;
 - Java ME (Micro Edition): Usada em dispositivos móveis e softwares embarcados;
 - Java Card: Utilizada em dispositivos móveis com limitações de recursos (Smart-cards);
 - Java FX: Utilizada em aplicações multimídia para desktop ou web.
- ✓ Para executar uma aplicação em Java sabemos que precisamos da Java Virtual Machine (JVM). A JVM fica dentro de um recurso chamado Java Runtime Environment (JRE)

Ambiente de programação:

- ✓ Basicamente, é possível desenvolver em Java usando apenas dois componentes principais: um editor de texto comum e o JDK.

➤ Eclipse;	➤ Jbuilder;	➤ BlueJ;
➤ NetBeans;	➤ Jdeveloper;	➤ IntelliJIDEA;

Conceituando Orientação a Objetos

Programação Orientação a Objetos (POO) é uma abordagem que procura explorar a intuição.

Classe:

- ✓ Agrupamento de objetos similares.
- ✓ Todo objeto é uma instância de uma Classe.
- ✓ Os objetos representados por determinada classe diferenciam-se entre si pelos valores de seus atributos.
- ✓ Conjunto de objetos que possuem propriedades semelhantes (Atributos), o mesmo comportamento (Métodos), os mesmos relacionamentos com outros objetos e a mesma semântica.
- ✓ Modificadores de acesso p/ Classes:
 - Default (pacote): classes sem modificador de acesso explícito, só podem ser instanciadas dentro da própria classe ou por classes do mesmo pacote.
Ex: `class NomeClasse`
 - Public : Classes public podem ser instanciadas por qualquer objeto livremente.
Ex: `public class NomeClasse`

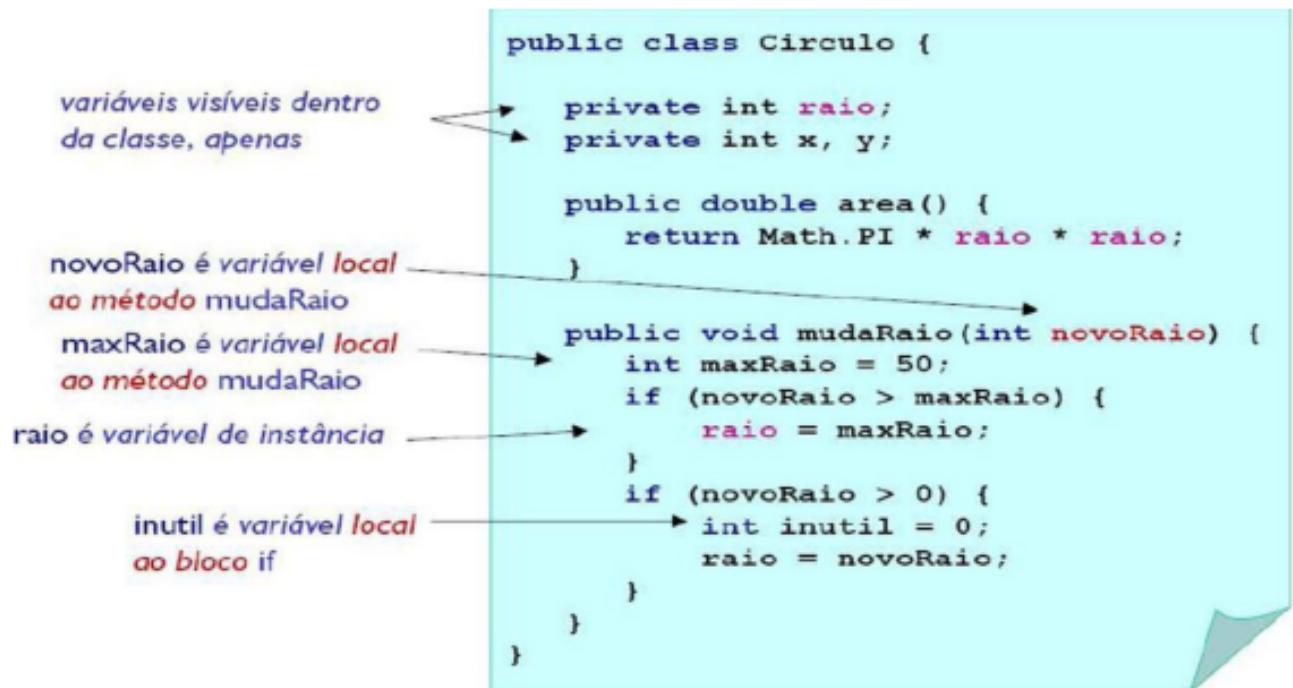
Objeto:

- ✓ É Representação computacional análoga de algo do mundo real (Concreto, Abstrato)
- ✓ É Dinâmico: é criado por alguém, tem uma vida, e morre ou é morto por alguém. Assim, durante a execução do sistema, os objetos podem:
 - Ser construídos; executar ações; ser destruídos; tornar-se inacessíveis;
- ✓ Os objetos que compartilham uma mesma interface, ou seja, respondem as mesmas mensagens, são agrupados em classes.
- ✓ É uma instância de uma classe: Instanciar um objeto é criar seu espaço de memória e repassar um ponteiro para ele.
Ex: `TipoClasse nome = new ConstrutorTipoClasse();`
`Carro ferrari = new Carro();`

Atributo:

- ✓ Representam um conjunto de informações, ou seja, elementos de dados que caracterizam um objeto;
- ✓ Descrevem as informações que ficam escondidas em um objeto para serem exclusivamente manipuladas pelas operações daquele objeto;
- ✓ Cada objeto possui seu próprio conjunto de atributos
- ✓ Modificadores de acesso p/ Atributos:
<visibilidade> - é o modificador de acesso para o atributo, podendo ser:
 - Public: poderá ser acessado por qualquer classe.

- Private: só poderá ser acessado dentro da mesma classe.
- Default (Package): só poderá ser acessado dentro da própria classe ou por classes do mesmo pacote.
- Protected: só poderá ser acessado por classes do mesmo pacote ou por subclasses (veremos isto em herança).



Métodos:

- ✓ São procedimentos definidos e declarados que atuam sobre um objeto ou sobre uma classe de objetos;
- ✓ Métodos são invocados por Mensagens;
- ✓ Cada objeto possui seu próprio conjunto de métodos;
- ✓ MÉTODOS (MODIFICADORES DE ACESSO P/ MÉTODOS):
 <visibilidade> - é o modificador de acesso para o método, podendo ser:
 - Public: o método poderá ser acessado por qualquer classe.
 - Private: o método só poderá ser acessado dentro da mesma classe.
 - Default (Package): o método só poderá ser acessado dentro da própria classe ou por classes do mesmo pacote.
 - Protected: o método só poderá ser acessado por classes do mesmo pacote ou por subclasses (veremos isto em herança).
 - <static> : modificador que define se o método é de classe (com static) ou de instância (sem static). Métodos de classe podem ser chamados sem a especificação de um objeto.

```

<visibilidade> <static> <tipo_retorno> <nome_metodo>(<parametros>) {
    // corpo ou escopo do metodo
}

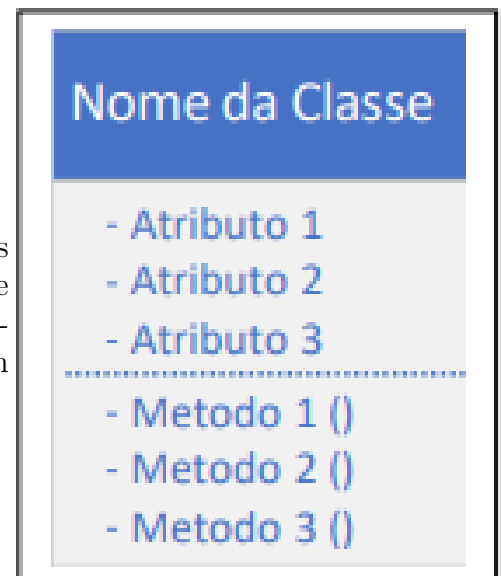
```

Objetos e Classes:

CLASSE CARRO		OBJETO CARRO A	OBJETO CARRO B
Atributos de objeto	Marca	Ford	Mitsubishi
	Modelo	Fiesta	L-200
	Cor	branco	azul royal
	Combustível	gasolina	diesel
Métodos	ligar		
	acelerar		
	frear		

Diagrama de Classe:

Representa a estrutura estática do sistema, mostrando as classes, seus atributos, métodos e os relacionamentos entre elas. Ele é essencial para a modelagem de sistemas orientados a objetos, pois define como as classes se organizam e interagem.



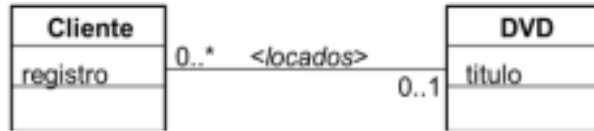
Componentes

- **Classes:** Representadas por retângulos divididos em três partes (nome, atributos e métodos).
- **Atributos:** Definem as características da classe. Cada atributo tem um tipo de dado (int, double, String).
- **Métodos:** Representam as operações que a classe pode realizar. No diagrama de classe, eles são apenas declarados, sem detalhar a implementação;
- **Visibilidade:** Define quem pode acessar os atributos e métodos de uma classe:

- Público (+): Acesso permitido por qualquer classe.
- Protected (#): Acesso restrito à própria classe e suas subclasses.
- Private (-): Acesso permitido apenas à própria classe.

• **Relacionamentos entre Classes:**

- **Associacao:** Vinculo entre duas classes.



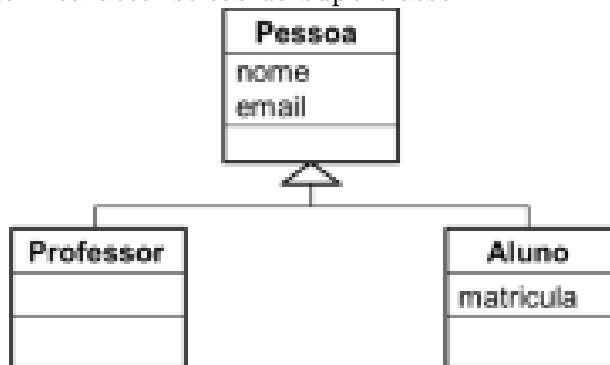
- **Agregacao:** Tipo de associacao que indica uma relacao "todo-parte".



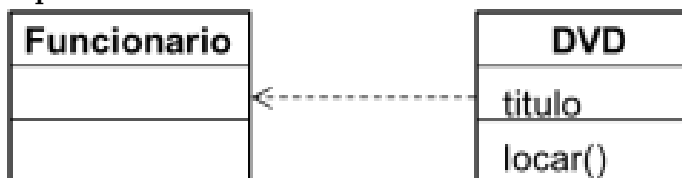
- **Composicao:** Tipo mais forte de agregacao onde a parte nao existe sem o todo.



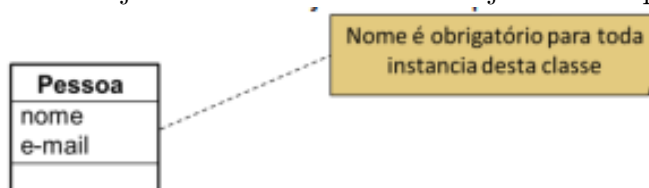
- **Heranca:** Define uma relação de generalização/especialização, onde subclasses herdam características da superclasse.



- **Dependência:** Relacionamento fraco onde uma classe depende de outra.



- **Notas:** Objetivo é informar como o objeto se comporta.



- **Multiplicidade:** Descrevem como as entidades se interagem entre si (Cardinalidade). Os relacionamentos podem ser de vários tipos, como:
 - ✓ Um para Um (1:1): Cada instância de uma entidade A está associada a uma única instância de uma entidade B.
 - ✓ Um para Muitos (1): Uma instância de uma entidade A pode estar associada a várias instâncias de uma entidade B, mas cada instância de B está associada a apenas uma de A.
 - ✓ Muitos para Muitos (N): Várias instâncias de uma entidade A podem estar associadas a várias instâncias de uma entidade B.

MULTIPLICIDADE	
0..1	No máximo um. Indica que os objetos da classe associada não precisam obrigatoriamente estar relacionados.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
0..*	Muitos. Indica que pode haver muitos objetos da classe envolvidos no relacionamento.
1..*	Um ou muitos. Indica que há pelo menos um objeto envolvido no relacionamento.
3..5	Valores específicos.

Tabela 1: Significados das multiplicidades em UML

Exemplo:

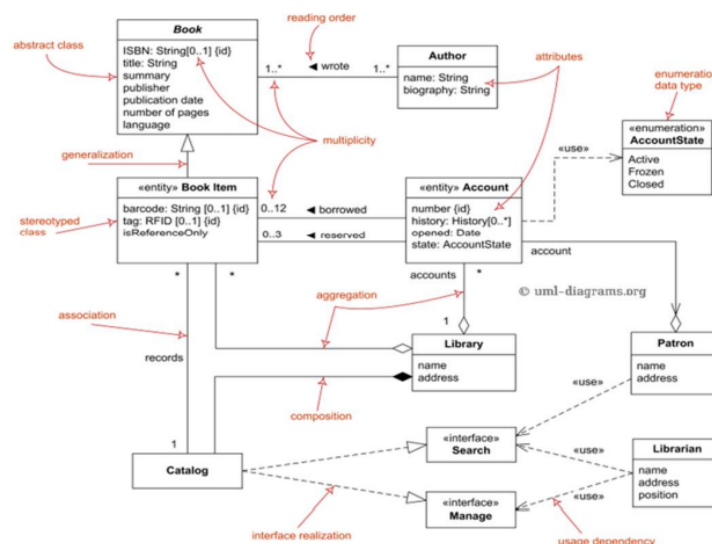
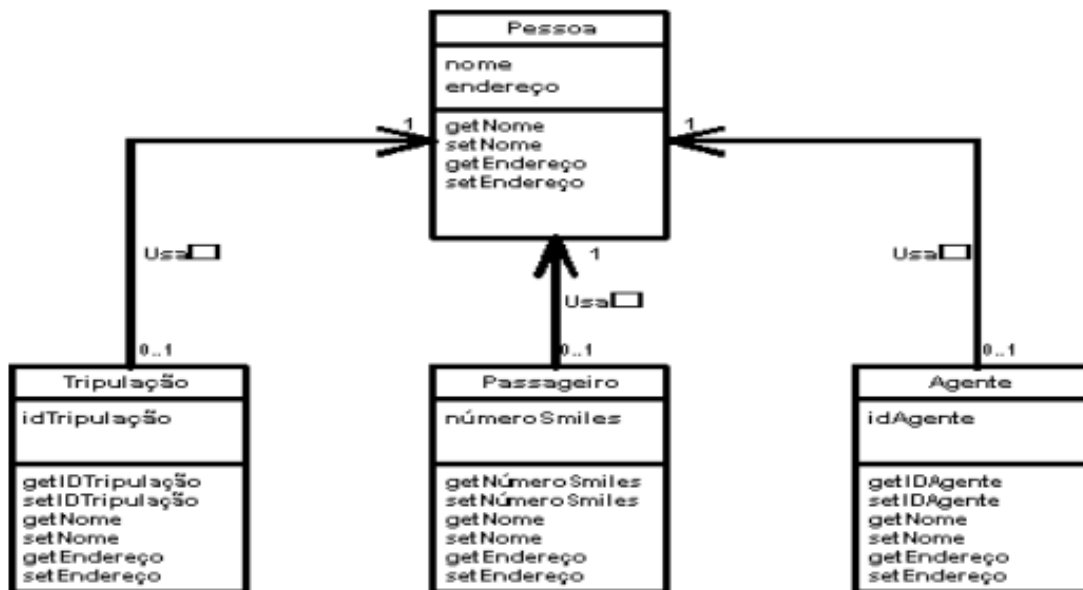


Figura 3: Diagrama UML representando classes, interfaces, enumerações e relacionamentos em um sistema de biblioteca.

Herança:

- ✓ Mecanismo que permite que características comuns a diversas classes sejam colocadas em uma classe base, ou superclasse.
- ✓ As propriedades da superclasse não precisam ser repetidas em cada subclasse.
- ✓ Por exemplo, JanelaRolante e JanelaFixa são subclasses de Janela. Elas herdam as propriedades de Janela;



Criando Projetos em Java

1. Criar novo projeto Java

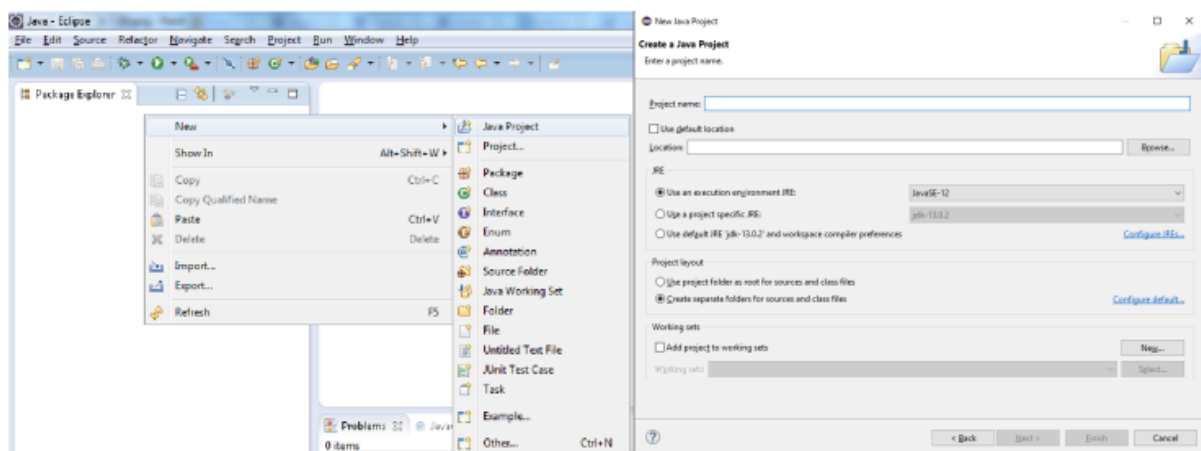


Figura 4: Criação de um novo projeto Java no Eclipse

2. Criar novo pacote e classe

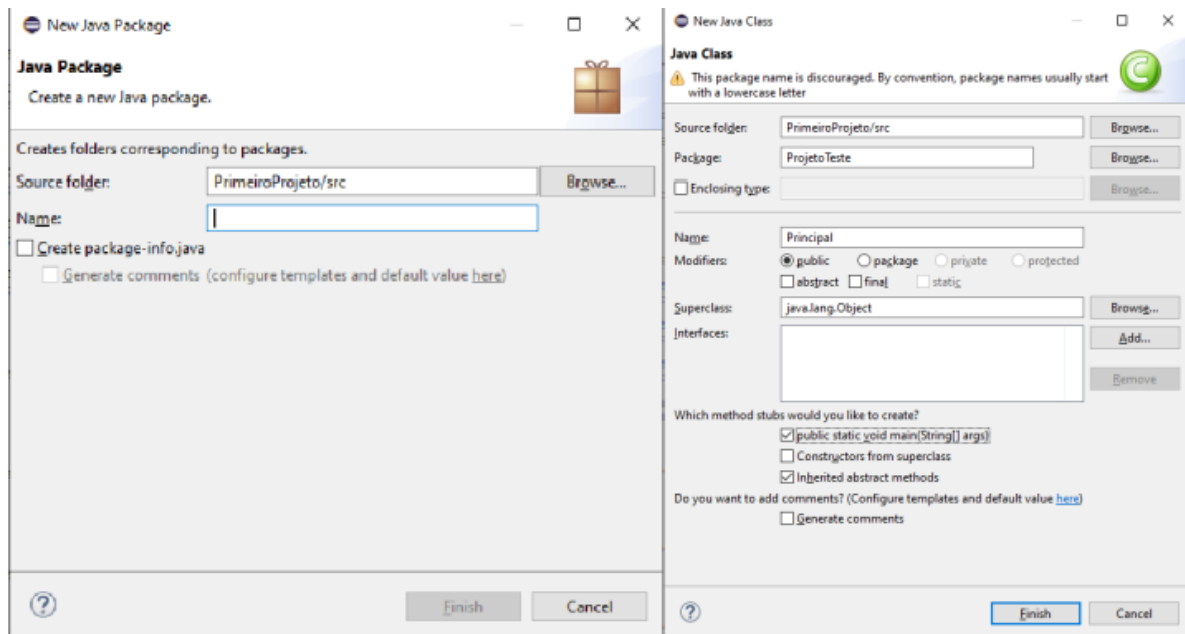


Figura 5: Criação do pacote e da classe Java

3. Código da classe Principal

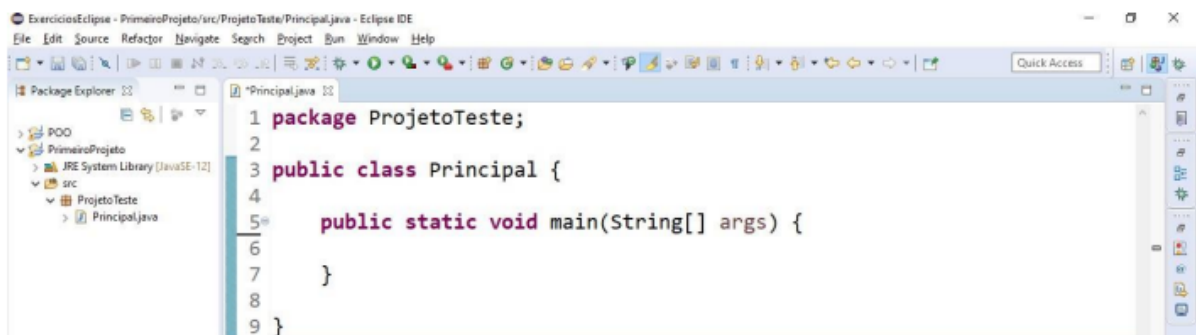


Figura 6: Código-fonte gerado na classe Principal.java

Fundamentos de Java

Tipos de Dados Primitivos

O que é: Tipos primitivos são os blocos básicos de dados em Java. Eles representam valores simples e são armazenados diretamente na memória.

Para que serve: Servem para armazenar valores básicos como números, caracteres e valores booleanos, formando a base para operações em programas Java.

Como usa:

Tipo	Bits	Intervalo (Inferior)	Intervalo (Superior)	Exemplo de Uso
byte	8	-128	127	byte idade = 30;
short	16	-32,768	32,767	short ano = 2023;
int	32	-2^{31}	$2^{31}-1$	int populacao = 210000000;
long	64	-2^{63}	$2^{63}-1$	long distancia = 15000000000L;
float	32	$\pm 1.4E-45$	$\pm 3.4028235E+38$	float preco = 19.99f;
double	64	$\pm 4.9E-324$	$\pm 1.7976931348623157E+308$	double pi = 3.141592653589793;
char	16	\u0000 (0)	\uffff (65,535)	char letra = 'A';
boolean	1	-	-	boolean ativo = true;

Estruturas Condicionais

O que é: Estruturas que permitem controlar o fluxo de execução do programa com base em condições.

Para que serve: Tomar decisões no código, executando diferentes blocos conforme condições específicas.

Como usa:

if-else

```

1 int idade = 18;
2
3 if (idade >= 18) {
4     System.out.println("Maior de idade");
5 } else {
6     System.out.println("Menor de idade");
7 }

```

Listing 5: Exemplo de if-else em Java

switch

```

1 int diaSemana = 3;
2 String nomeDia;
3
4 switch (diaSemana) {
5     case 1:
6         nomeDia = "Domingo";
7         break;
8     case 2:
9         nomeDia = "Segunda";
10        break;
11    case 3:
12        nomeDia = "Terca";
13        break;
14    // outros casos...
15    default:
16        nomeDia = "Dia invalido";
17 }
18
19 System.out.println(nomeDia); // Imprime "Terca"

```

Listing 6: Exemplo de switch-case em Java

Estruturas de Repetição

O que é: Estruturas que permitem executar um bloco de código repetidamente.

Para que serve: Automatizar tarefas repetitivas e processar coleções de dados.

Como usa:

for tradicional

```
1 for (int i = 0; i < 5; i++) {  
2     System.out.println("Iteração: " + i);  
3 }
```

Listing 7: Exemplo de laço for tradicional em Java

for aprimorado (for-each)

```
1 String[] frutas = {"Maçã", "Banana", "Laranja"};  
2  
3 for (String fruta : frutas) {  
4     System.out.println(fruta);  
5 }
```

Listing 8: Exemplo de laço for-each (for aprimorado) em Java

while

```
1 int contador = 0;  
2  
3 while (contador < 3) {  
4     System.out.println("Contagem: " + contador);  
5     contador++;  
6 }
```

Listing 9: Exemplo de laço while em Java

do-while

```
1 int contador = 0;  
2  
3 do {  
4     System.out.println("Contagem: " + contador);  
5     contador++;  
6 } while (contador < 3);
```

Listing 10: Exemplo de laço do-while em Java

Matrizes e Arrays

O que é: Estruturas que armazenam coleções de elementos do mesmo tipo.

Para que serve: Organizar e manipular conjuntos de dados relacionados.

Como usa:

Array unidimensional

```

1 // Declaração e inicialização
2 int[] numeros = new int[3];
3 numeros[0] = 10;
4 numeros[1] = 20;
5 numeros[2] = 30;
6
7 // Forma simplificada
8 int[] outrosNumeros = {40, 50, 60};
9
10 // Acessando elementos
11 System.out.println(numeros[1]); // 20

```

Listing 11: Exemplo de uso de arrays em Java

Matriz (array bidimensional)

```

1 // Declaração
2 int[][] matriz = new int[2][3];
3
4 // Inicialização
5 matriz[0][0] = 1;
6 matriz[0][1] = 2;
7 matriz[0][2] = 3;
8 matriz[1][0] = 4;
9 matriz[1][1] = 5;
10 matriz[1][2] = 6;
11
12 // Forma simplificada
13 int[][] outraMatriz = {{1, 2, 3}, {4, 5, 6}};
14
15 // Acessando elementos
16 System.out.println(matriz[1][2]); // 6

```

Listing 12: Exemplo de uso de matrizes (arrays 2D) em Java

Programação Orientada a Objetos

Pacotes, Classe e Objeto

O que é:

- **Pacote:** Agrupamento lógico de classes relacionadas
- **Classe:** Modelo/planta que define atributos e métodos
- **Objeto:** Instância concreta de uma classe

Para que serve: Organizar código e modelar entidades do mundo real.

Como usa:

```

1 // Definição de pacote (deve ser a primeira linha do arquivo)
2 package com.exemplo.veiculos;
3
4 // Definição de classe
5 public class Carro {
6     // Atributos
7     String modelo;
8     int ano;
9
10    // Método
11    public void acelerar() {
12        System.out.println("Carro acelerando...");
13    }
14 }
15
16 // Criando objeto
17 public class Main {
18     public static void main(String[] args) {
19         Carro meuCarro = new Carro();
20         meuCarro.modelo = "Fusca";
21         meuCarro.ano = 1976;
22         meuCarro.acelerar();
23     }
24 }

```

Atributos e Métodos

O que é:

- **Atributos:** Características/variáveis da classe
- **Métodos:** Ações/funções da classe

Para que serve: Definir o estado e comportamento dos objetos.

Como usa:

```

1 public class ContaBancaria {
2     // Atributos
3     private String titular;
4     private double saldo;
5
6     // Método construtor
7     public ContaBancaria(String titular, double saldoInicial) {
8         this.titular = titular;
9         this.saldo = saldoInicial;
10    }
11
12    // Métodos
13    public void depositar(double valor) {
14        saldo += valor;

```

```

15     }
16
17     public boolean sacar(double valor) {
18         if (valor <= saldo) {
19             saldo -= valor;
20             return true;
21         }
22         return false;
23     }
24
25     public double consultarSaldo() {
26         return saldo;
27     }
28 }

```

Encapsulamento

O que é: Princípio de ocultar detalhes internos e proteger dados.

Para que serve: Manter integridade dos dados e facilitar manutenção.

Como usa:

```

1 public class Pessoa {
2     // Atributos privados
3     private String nome;
4     private int idade;
5
6     // Métodos públicos para acesso (getters e setters)
7     public String getNome() {
8         return nome;
9     }
10
11     public void setNome(String nome) {
12         if (nome != null && !nome.isEmpty()) {
13             this.nome = nome;
14         }
15     }
16
17     public int getIdade() {
18         return idade;
19     }
20
21     public void setIdade(int idade) {
22         if (idade >= 0) {
23             this.idade = idade;
24         }
25     }
26 }

```

Herança (super e this)

O que é: Mecanismo onde uma classe herda atributos e métodos de outra.

Para que serve: Promover reutilização de código e criar hierarquias.

Como usa:

```
1 // Classe pai
2 public class Veiculo {
3     protected String marca;
4
5     public Veiculo(String marca) {
6         this.marca = marca;
7     }
8
9     public void mover() {
10        System.out.println("Veículo se movendo");
11    }
12 }
13
14 // Classe filha
15 public class Carro extends Veiculo {
16     private int portas;
17
18     public Carro(String marca, int portas) {
19         super(marca); // Chama construtor da classe pai
20         this.portas = portas;
21     }
22
23     @Override
24     public void mover() {
25         System.out.println("Carro da marca " + marca + " está
26             andando");
27     }
28
29     public void mostrarDetalhes() {
30         System.out.println("Marca: " + super.marca); // Acessando
31             atributo da classe pai
32         System.out.println("Portas: " + this.portas); // Acessando
33             atributo da própria classe
34     }
35 }
```

Herança Múltipla com Interfaces

O que é: Java não permite herança múltipla de classes, mas permite implementar múltiplas interfaces.

Para que serve: Definir contratos que classes podem implementar, permitindo polimorfismo.

Como usa:

```

1 // Interface 1
2 public interface Autenticavel {
3     boolean autenticar(String senha);
4 }
5
6 // Interface 2
7 public interface Auditar {
8     void gerarLog();
9 }
10
11 // Classe implementando múltiplas interfaces
12 public class Usuario implements Autenticavel, Auditar {
13     private String login;
14     private String senha;
15
16     @Override
17     public boolean autenticar(String senha) {
18         return this.senha.equals(senha);
19     }
20
21     @Override
22     public void gerarLog() {
23         System.out.println("Log de acesso do usuário: " + login);
24     }
25 }

```

Polimorfismo (sobrecarga e sobrescrita)

O que é: Capacidade de um objeto ser referenciado de várias formas.

Para que serve: Flexibilidade e extensibilidade no código.

Como usa:

Sobrecarga (overload)

```

1 public class Calculadora {
2     // Métodos com mesmo nome mas parâmetros diferentes
3     public int somar(int a, int b) {
4         return a + b;
5     }
6
7     public double somar(double a, double b) {
8         return a + b;
9     }
10
11     public int somar(int a, int b, int c) {
12         return a + b + c;
13     }
14 }

```

Sobrescrita (override)

```
1 public class Animal {
2     public void emitirSom() {
3         System.out.println("Som genérico de animal");
4     }
5 }
6
7 public class Cachorro extends Animal {
8     @Override
9     public void emitirSom() {
10        System.out.println("Au au!");
11    }
12 }
13
14 public class Gato extends Animal {
15     @Override
16     public void emitirSom() {
17        System.out.println("Miau!");
18    }
19 }
```

Abstração

O que é: Focar nos aspectos essenciais, ignorando detalhes irrelevantes. Não há código específico para abstração. A abstração é um princípio que se manifesta no design das classes. Em vez de pensar em todos os detalhes de um carro (motor, transmissão, etc.)

Para que serve: Simplificar sistemas complexos.

Como usa:

Podemos abstrair para os aspectos relevantes ao nosso sistema

```
1
2 public class Carro {
3     private String modelo;
4     private int velocidadeAtual;
5
6     public void acelerar() { /* ... */ }
7     public void frear() { /* ... */ }
8 }
```

Interface

O que é: Contrato que define métodos que uma classe deve implementar.

Para que serve: Estabelecer comportamentos comuns sem herança.

Como usa:

```
1 public interface FormaGeometrica {
2     double calcularArea();
3     double calcularPerimetro();
4 }
```

```

4 }
5
6 public class Retangulo implements FormaGeometrica {
7     private double largura;
8     private double altura;
9
10    public Retangulo(double largura, double altura) {
11        this.largura = largura;
12        this.altura = altura;
13    }
14
15    @Override
16    public double calcularArea() {
17        return largura * altura;
18    }
19
20    @Override
21    public double calcularPerimetro() {
22        return 2 * (largura + altura);
23    }
24 }
25
26 public class Circulo implements FormaGeometrica {
27     private double raio;
28
29     public Circulo(double raio) {
30         this.raio = raio;
31     }
32
33     @Override
34     public double calcularArea() {
35         return Math.PI * raio * raio;
36     }
37
38     @Override
39     public double calcularPerimetro() {
40         return 2 * Math.PI * raio;
41     }
42 }

```

Classe Abstrata

O que é: Classe que não pode ser instanciada, servindo como modelo.

Para que serve: Fornecer implementação comum para subclasses.

Como usa:

```

1 public abstract class Funcionario {
2     private String nome;
3     private String matricula;
4

```



```

5     public Funcionario(String nome, String matricula) {
6         this.nome = nome;
7         this.matricula = matricula;
8     }
9
10    // Método abstrato - deve ser implementado pelas subclasses
11    public abstract double calcularSalario();
12
13    // Método concreto
14    public String getNome() {
15        return nome;
16    }
17 }
18
19 public class Gerente extends Funcionario {
20     private double salarioBase;
21     private double bonus;
22
23     public Gerente(String nome, String matricula, double
24         salarioBase, double bonus) {
25         super(nome, matricula);
26         this.salarioBase = salarioBase;
27         this.bonus = bonus;
28     }
29
30     @Override
31     public double calcularSalario() {
32         return salarioBase + bonus;
33     }
34
35 public class Vendedor extends Funcionario {
36     private double comissao;
37     private int vendas;
38
39     public Vendedor(String nome, String matricula, double comissao,
40         int vendas) {
41         super(nome, matricula);
42         this.comissao = comissao;
43         this.vendas = vendas;
44     }
45
46     @Override
47     public double calcularSalario() {
48         return comissao * vendas;
49     }

```

Enum

O que é: Tipo especial que representa um conjunto de constantes.

Para que serve: Melhorar legibilidade e segurança com valores pré-definidos.

Como usa:

```
1 public enum DiaSemana {
2     SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO;
3 }
4
5 public class TesteEnum {
6     public static void main(String[] args) {
7         DiaSemana hoje = DiaSemana.QUARTA;
8
9         switch (hoje) {
10             case SEGUNDA:
11                 System.out.println("Dia de começar a semana!");
12                 break;
13             case QUARTA:
14                 System.out.println("Metade da semana!");
15                 break;
16             case SEXTA:
17                 System.out.println("Sextou!");
18                 break;
19             default:
20                 System.out.println("Outro dia da semana");
21         }
22
23         // Iterando sobre os valores do enum
24         for (DiaSemana dia : DiaSemana.values()) {
25             System.out.println(dia);
26         }
27     }
28 }
```

Associação, Agregação e Composição

O que é: Tipos de relacionamentos entre classes:

- **Associação:** Relação simples entre objetos
- **Agregação:** Relação "tem-um" onde o todo pode existir sem as partes
- **Composição:** Relação "tem-um" onde o todo controla o ciclo de vida das partes

Para que serve: Modelar relacionamentos entre entidades.

Como usa:

Associação

```
1 public class Professor {
2     private String nome;
3
4     public Professor(String nome) {
5         this.nome = nome;
6     }
7 }
```

```

6     }
7
8     // getters e setters
9 }
10
11 public class Disciplina {
12     private String nome;
13     private Professor professor; // Associação
14
15     public Disciplina(String nome, Professor professor) {
16         this.nome = nome;
17         this.professor = professor;
18     }
19
20     // getters e setters
21 }

```

Agregação

```

1 public class Departamento {
2     private String nome;
3     private List<Professor> professores; // Agregação
4
5     public Departamento(String nome) {
6         this.nome = nome;
7         this.professores = new ArrayList<>();
8     }
9
10    public void adicionarProfessor(Professor professor) {
11        professores.add(professor);
12    }
13
14    // outros métodos
15 }

```

Composição

```

1 public class Carro {
2     private Motor motor; // Composição
3
4     public Carro() {
5         this.motor = new Motor(); // Motor criado junto com o Carro
6     }
7
8     // Quando o Carro é destruído, o Motor também é
9 }
10
11 public class Motor {
12     // Atributos e métodos do motor

```

Modificadores de Acesso

O que é: Palavras-chave que controlam a visibilidade de classes, atributos e métodos.

Para que serve: Implementar encapsulamento e controlar acesso.

Como usa:

Modificador	Classe	Pacote	Subclasse	Todos
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
default (sem modificador)	Sim	Sim	Não	Não
private	Sim	Não	Não	Não

```

1 public class ExemploAcesso {
2     public int publico;           // Acessível de qualquer lugar
3     protected int protegido;     // Acessível no pacote e por
        subclasses
4     int pacote;                  // Acessível apenas no pacote (
        default)
5     private int privado;         // Acessível apenas na classe
6
7     public void metodoPublico() {
8         // Todos os atributos são acessíveis aqui
9         this.privado = 10;
10    }
11
12    private void metodoPrivado() {
13        // Método acessível apenas dentro da classe
14    }
15 }
```

Conceitos Avançados e Utilitários

Leitura de arquivos .csv com BufferedReader

O que é: Técnica para ler dados de arquivos CSV (Comma-Separated Values).

Para que serve: Importar dados de planilhas ou sistemas externos para processamento em Java.

Como usa:

```

1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.io.IOException;
4
5 public class LeitorCSV {
6     public static void main(String[] args) {
```

```

7      String arquivoCSV = "dados.csv";
8      String linha = "";
9      String separador = ",";
10
11     try (BufferedReader br = new BufferedReader(new FileReader(
12         arquivoCSV))) {
13         // Lê cada linha do arquivo
14         while ((linha = br.readLine()) != null) {
15             // Divide a linha usando o separador
16             String[] dados = linha.split(separador);
17
18             // Processa os dados
19             for (String dado : dados) {
20                 System.out.print(dado + " | ");
21             }
22             System.out.println();
23         } catch (IOException e) {
24             e.printStackTrace();
25         }
26     }
27 }

```

Listing 13: Exemplo de leitura de arquivo CSV em Java

Tratamento de exceções com try-catch

O que é: Mecanismo para lidar com erros e situações excepcionais durante a execução.

Para que serve: Evitar que o programa termine abruptamente quando ocorrem erros, permitindo tratamento adequado.

Como usa:

```

1 public class Divisao {
2     public static void main(String[] args) {
3         int a = 10;
4         int b = 0;
5
6         try {
7             // Código que pode lançar exceção
8             int resultado = a / b;
9             System.out.println("Resultado: " + resultado);
10        } catch (ArithmeticException e) {
11            // Tratamento da exceção
12            System.out.println("Erro: Divisão por zero!");
13            System.out.println("Mensagem de erro: " + e.getMessage());
14        } finally {
15            // Bloco opcional que sempre executa
16            System.out.println("Fim do cálculo.");
17        }
18    }

```

19 }

Listing 14: Exemplo de try-catch em Java

Coleções Java

O que é: Estruturas de dados prontas para armazenar e manipular grupos de objetos.

Para que serve: Trabalhar com conjuntos de dados de forma eficiente.

Como usa:

ArrayList

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class ExemploArrayList {
5     public static void main(String[] args) {
6         List<String> nomes = new ArrayList<>();
7
8         // Adicionando elementos
9         nomes.add("Alice");
10        nomes.add("Bob");
11        nomes.add("Charlie");
12
13        // Acessando elementos
14        System.out.println(nomes.get(1)); // Bob
15
16        // Iterando
17        for (String nome : nomes) {
18            System.out.println(nome);
19        }
20
21        // Removendo
22        nomes.remove("Bob");
23    }
24 }
```

HashMap

```
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class ExemploHashMap {
5     public static void main(String[] args) {
6         Map<String, Integer> idades = new HashMap<>();
7
8         // Adicionando pares chave-valor
9         idades.put("Alice", 25);
10        idades.put("Bob", 30);
11        idades.put("Charlie", 28);
```

```

12
13 // Acessando valores
14 System.out.println(idades.get("Bob")); // 30
15
16 // Verificando existência
17 if (idades.containsKey("Alice")) {
18     System.out.println("Alice está no mapa");
19 }
20
21 // Iterando
22 for (Map.Entry<String, Integer> entrada : idades.entrySet()
23     ) {
24     System.out.println(entrada.getKey() + ": " + entrada.
25         getValue());
26 }

```

HashSet

```

1 import java.util.HashSet;
2 import java.util.Set;
3
4 public class ExemploHashSet {
5     public static void main(String[] args) {
6         Set<String> conjunto = new HashSet<>();
7
8         // Adicionando elementos (não permite duplicatas)
9         conjunto.add("Maçã");
10        conjunto.add("Banana");
11        conjunto.add("Maçã"); // Não será adicionado novamente
12
13        System.out.println(conjunto.size()); // 2
14
15        // Verificando existência
16        if (conjunto.contains("Banana")) {
17            System.out.println("Banana está no conjunto");
18        }
19
20        // Iterando
21        for (String fruta : conjunto) {
22            System.out.println(fruta);
23        }
24    }
25 }

```

Interface Gráfica com Java Swing

O que é: Conjunto de componentes para criar interfaces gráficas.

Para que serve: Desenvolver aplicações desktop com interface amigável.

Como usa:

JFrame básico

```
1 import javax.swing.JButton;
2 import javax.swing.JFrame;
3 import javax.swing.JLabel;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6
7 public class MinhaJanela extends JFrame {
8     public MinhaJanela() {
9         // Configurações básicas da janela
10        setTitle("Minha Primeira Janela");
11        setSize(300, 200);
12        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13        setLayout(null);
14
15        // Criando componentes
16        JLabel label = new JLabel("Olá, Swing!");
17        label.setBounds(100, 30, 100, 20);
18
19        JButton botao = new JButton("Clique Aqui");
20        botao.setBounds(90, 80, 120, 30);
21
22        // Adicionando ação ao botão
23        botao.addActionListener(new ActionListener() {
24            @Override
25            public void actionPerformed(ActionEvent e) {
26                label.setText("Botão clicado!");
27            }
28        });
29
30        // Adicionando componentes a janela
31        add(label);
32        add(botao);
33    }
34
35    public static void main(String[] args) {
36        MinhaJanela janela = new MinhaJanela();
37        janela.setVisible(true);
38    }
39 }
```

Exemplo com vários componentes

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4
```



```

5 public class Formulario extends JFrame {
6     private JTextField campoNome;
7     private JTextArea areaTexto;
8
9     public Formulario() {
10         setTitle("Formulário");
11         setSize(400, 300);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         setLayout(new BorderLayout());
14
15         // Painel superior
16         JPanel painelSuperior = new JPanel();
17         painelSuperior.add(new JLabel("Nome:"));
18         campoNome = new JTextField(20);
19         painelSuperior.add(campoNome);
20
21         // Área central
22         areaTexto = new JTextArea();
23         JScrollPane scroll = new JScrollPane(areaTexto);
24
25         // Painel inferior
26         JPanel painelInferior = new JPanel();
27         JButton botaoEnviar = new JButton("Enviar");
28         botaoEnviar.addActionListener(this::acaoEnviar);
29         painelInferior.add(botaoEnviar);
30
31         // Adicionando componentes a janela
32         add(painelSuperior, BorderLayout.NORTH);
33         add(scroll, BorderLayout.CENTER);
34         add(painelInferior, BorderLayout.SOUTH);
35     }
36
37     private void acaoEnviar(ActionEvent e) {
38         String nome = campoNome.getText();
39         areaTexto.append("Olá, " + nome + "!\n");
40         campoNome.setText("");
41     }
42
43     public static void main(String[] args) {
44         SwingUtilities.invokeLater(() -> {
45             Formulario formulario = new Formulario();
46             formulario.setVisible(true);
47         });
48     }
49 }

```

Exercício em Java

```

1 package Aulas;
2

```

```

3 import java.util.Scanner;    // Importacao da Biblioteca Scanner
4
5 public class Principal {
6
7     private static Scanner Teclado;    // Declaracao da Variavel
        Teclado do Tipo Scanner
8
9     public static void main(String[] args) {
10
11         float Valor_1, Valor_2, Result = 0;    // Declaracao de
            Variaveis
12         Teclado = new Scanner(System.in);    // Ler Valor Digitado
13
14         System.out.print("Digite o primeiro Valor: ");    // Escrever
            na Tela
15         Valor_1 = Teclado.nextFloat();    // Armazena
            valor digitado
16
17         System.out.print("Digite o segundo Valor: ");
18         Valor_2 = Teclado.nextFloat();
19
20         Result = Valor_1 + Valor_2;
21         System.out.printf("Result Soma = %f", Result);    // Exibir
            Resultado
22     }
23 }

```

Listing 15: Leitura de valores com Scanner e soma em Java

```

1 package Aulas;
2
3 public class If_Else_ElseIf {
4     public static void main(String[] args) {
5         int Op = 2;
6
7         if (Op == 1) {    // SE
8             System.out.print("Condicao If");
9         } else if (Op == 2) {    // SE NAO SE
10             System.out.print("Condicao ElseIf");
11         } else {    // SE NAO
12             System.out.print("Condicao Else");
13         }
14     }
15 }

```

Listing 16: Condicional if-else-if

```

1 package Aulas;
2
3 public class For_While_DoWhile {
4     public static void main(String[] args) {
5         int i;

```

```

6      System.out.print("Condicao For Incremento \n");
7      for (i = 0; i < 10; i++) {
8          System.out.printf("%d \t", i);
9      }
10
11      System.out.print("\nCondicao For Decremento \n");
12      for (i = 9; i >= 0; i--) {
13          System.out.printf("%d \t", i);
14      }
15  }
16 }
17 }

```

Listing 17: Estrutura de repeticao for

```

1 package Aulas;
2
3 public class For_While_DoWhile {
4     public static void main(String[] args) {
5
6         System.out.print("\nCondicao While \n");
7         int j = 0;
8         while (j != 10) {
9             System.out.printf("%d \t", j);
10            j++;
11        }
12
13        System.out.print("\nCondicao DoWhile \n");
14        int x = 2;
15        System.out.printf("%d \t", x);
16        do {
17            System.out.printf("%d \t", x);
18            x--;
19        } while (x != 0);
20    }
21 }

```

Listing 18: Estruturas While e Do While

```

1 package Aulas;
2 import java.util.Scanner;
3
4 public class SwitchCase {
5     private static Scanner Entrada;
6
7     public static void main(String[] args) {
8         Entrada = new Scanner(System.in);
9
10        System.out.print("Entre com um numero entre 1 e 4: ");
11        //int Opcao = Entrada.nextInt();
12        char Opcao = Entrada.next().charAt(0);
13    }

```

```

14     switch (Opcao) {
15         case '+':
16             //case 1:
17             System.out.println("Voce escolheu 1 ou +");
18             break;
19         case '-':
20             //case 2:
21             System.out.println("Voce escolheu 2 ou -");
22             break;
23         default:
24             System.out.println("Numero invalido");
25     }
26 }
27 }

```

Listing 19: Exemplo de estrutura Switch Case

```

1 package Aulas;
2
3 public class ArrayVetorMatriz {
4     public static void main(String[] args) {
5         //int[] a = new int[4];
6
7         //int[] b;
8         //b = new int[10];
9
10        //int[] r = new int[44], k = new int[23];
11
12        //int[] iniciaValores = {12,32,54,6,8,89,64,64,6};
13
14        int[] meuArray;
15        meuArray = new int[10];    //ALOCA MEMORIA PARA 10 INTEIROS
16
17        //INICIALIZA O PRIMEIRO ELEMENTO
18        meuArray[0] = 100;
19        meuArray[1] = 180;
20        meuArray[2] = 123;
21        meuArray[3] = 93;
22        meuArray[4] = 345;
23        meuArray[5] = 923;
24        meuArray[6] = 354;
25        meuArray[7] = 822;
26        meuArray[8] = 623;
27        meuArray[9] = 564;
28        //meuArray[10] = 564; //ESTOURA A PILHA POIS NAO EXISTE O
        //INDICE 10
29
30        System.out.println(meuArray[9]);
31
32        //Percorrer Vetor/Array
33        for(int i = 0; i <= 5; i++) {
34            meuArray[i] = 12; // Linhas de Comando

```

```

35     }
36 }
37 }

```

Listing 20: Exemplo de uso de Array (Vetor)

Bibliografia

- ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. *Fundamentos da programação de computadores: algoritmos, pascal, C/C++ (padrão ANSI) e java*. 3. ed. São Paulo: Pearson Education do Brasil, 2012. 569 p. ISBN 9788564574168.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *UML: guia do usuário*. Tradução de Fabio Freitas. 2. ed. revista e atualizada. Rio de Janeiro: Campus: Elsevier, 2006. ISBN 85-352-0562-4.
- MACHADO, Rodrigo Prestes; BERTAGNOLLI, Silvia de Castro; FRANCO, Márcia Islabão. *Desenvolvimento de software v. 3: programação de sistemas web orientada a objetos em Java*. Porto Alegre: Bookman, 2016. ISBN 9788582603710. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788582603710>. Acesso em: 13 nov. 2024.
- SIERRA, Kathy; BATES, Bert. *Use a cabeça! Java*. Tradução de Aldir Jose Coelho. 2. ed. Rio de Janeiro: Alta Books, 2010. ISBN 857608084-2.
- SILVA, Robson Soares. *Oracle database 10g Express edition: guia de instalação, configuração e administração com Implementação PL/SQL Relacional e Objeto-Relacional*. São Paulo: Érica, 2007. ISBN 9788536519456. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788536519456>. Acesso em: 13 nov. 2024.
- EDELWEISS, Nina; LIVI, Maria Aparecida Castro. *Algoritmos e programação com exemplos em Pascal e C*. Porto Alegre: Bookman, 2014. ISBN 9788582601907. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788582601907>. Acesso em: 13 nov. 2024.
- FOWLER, Martin. *UML essencial: um breve guia para linguagem-padrão de modelagem de objetos*. 3. ed. Porto Alegre: Bookman, 2011. ISBN 9788560031382. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788560031382>. Acesso em: 13 nov. 2024.
- LARMAN, Craig. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo*. 3. ed. Porto Alegre: Bookman, 2007. ISBN 9788577800476. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788577800476>. Acesso em: 13 nov. 2024.
- SCHILDT, Herbert. *Java para iniciantes: crie, compile e execute programas Java rapidamente*. 6. ed. Porto Alegre: Bookman, 2015. ISBN 9788582603376. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788582603376>. Acesso em: 13 nov. 2024.

- WAZLAWICK, Raul Sidnei. *Análise e design orientados a objetos para sistema de informação: modelagem com UML, OCL e IFML*. 3. ed. Rio de Janeiro: Elsevier, 2015. ISBN 9788595153653. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595153653/cfi/6/2!/4/2@0.00:0.00>. Acesso em: 13 nov. 2024.
- **PERIÓDICO:** ACM Transactions on Programming Languages and Systems. New York: Association for Computing Machinery, 2009-. ISSN: 0164-0925. Disponível em: <https://eds.s.ebscohost.com/eds/command/detail?vid=0&sid=eb26f77e-98fa-4137-940redis>. Qualis B1.
- **PERIÓDICO:** Numerical Algorithms. Basel: Springer Nature, 1991-. ISSN: 1017-1398. Disponível em: <https://www-springer-com.ez151.periodicos.capes.gov.br/journal/11075>. Qualis A2.
- **PERIÓDICO:** ACM Transactions on Algorithms. New York: Association for Computing Machinery, 2006-. ISSN: 1549-6325. Disponível em: <http://search.ebscohost.com/login.aspx?direct=true&db=iih&jid=22QW=pt-br&site=ehost-live>. Qualis B1.
- **PERIÓDICO:** Science of Computer Programming. Amsterdã: Elsevier, 1981-. ISSN: 0167-6423. Disponível em: <https://www-sciencedirect.ez151.periodicos.capes.gov.br/journal/science-of-computer-programming>. Qualis A4.