

PROGRAMAÇÃO ORIENTADA A OBJETOS

PROGRAMAÇÃO ORIENTADA A OBJETOS

Público-alvo: Estudantes do Centro Universitário FMU.

Objetivo: Esta apostila tem como objetivo conduzir os alunos recém-ingressos na graduação em Tech na transição do raciocínio lógico e estruturado para o Paradigma de Programação Orientada a Objetos (POO). O material foi desenhado de forma altamente didática, com linguagem clara, analogias do cotidiano e exemplos progressivos de código em Java.

Além de cobrir os pilares fundamentais da POO (abstração, encapsulamento, herança, polimorfismo e coleções), a apostila culmina na aplicação prática de persistência de dados utilizando um banco de dados moderno NoSQL (MongoDB), capacitando o estudante a desenvolver aplicações completas de nível básico a intermediário com operações de CRUD (Create, Read, Update, Delete) integradas. Toda a estrutura do material está alinhada com o cronograma de 66 horas semestrais e o plano de ensino oficial.

Lattes: <http://lattes.cnpq.br/8753112683856964>

Orcid: <https://orcid.org/0009-0002-7102-4993>

LLM utilizada: Gemini Flash – Agosto de 2026.

Unidades e Conteúdo

Unidade 1: Introdução ao Paradigma de POO e Configuração do Ambiente Java

Foco: Apresentar a disciplina, compreender a transição do pensamento estruturado para a orientação a objetos e configurar o ambiente prático.

Conteúdo:

- A História do Java, Versão Atual e sua Importância na Computação
- O que é o paradigma Orientado a Objetos e quais suas vantagens sobre a programação estruturada.
- A linguagem Java: JVM (Java Virtual Machine), JDK e JRE.
- Instalação da IDE (VS Code / IntelliJ / Eclipse) e execução do clássico "Hello, World!".

Unidade 2: Fundamentos da Linguagem Java — Entrada, Processamento e Saída

Foco: Dominar a sintaxe inicial do Java, manipulação de variáveis, operadores e comandos de leitura/escrita.

Conteúdo:

- Estrutura básica de um programa Java: o método `public static void main(String[] args)`.
- Tipos primitivos (`int`, `double`, `boolean`, `char`) e a classe `String`.
- Declaração de variáveis, casting de tipos e operadores aritméticos.
- Entrada de dados pelo console com a classe `Scanner` e saída com `System.out.println()`.

Unidade 3: Lógica de Controle — Estruturas Condicionais e Laços de Repetição

Foco: Controlar o fluxo de execução dos programas em Java através de estruturas de tomada de decisão e repetição.

Conteúdo:

- Condicionais simples, compostas e encadeadas: `if`, `else if`, `else`.
- Seleção múltipla com `switch-case`.
- Operadores lógicos (`&&`, `||`, `!`) e relacionais (`==`, `!=`, `>`, `<`).
- Estruturas de repetição: `while`, `do-while` e `for`.

Unidade 4: O Coração da POO — Classes, Objetos, Atributos, Métodos e Encapsulamento

Foco: Abstrair entidades do mundo real criando classes, instanciando objetos e protegendo dados através do encapsulamento.

Conteúdo:

- Conceito de Classe (molde/planta baixa) vs. Objeto (instância real).
- Definição de Atributos (características) e Métodos (comportamentos).
- Instanciação de objetos com new e papel dos métodos construtores.
- Encapsulamento: modificadores de acesso (private, public) e o uso de getters e setters.
- Membros estáticos (static): atributos e métodos da classe.

Unidade 5: Reuso e Hierarquia de Código — O Conceito de Herança

Foco: Reutilizar estruturas de código e criar hierarquias entre classes utilizando o mecanismo de herança em Java.

Conteúdo:

- O que é herança? Classe Pai (Superclasse) vs. Classe Filha (Subclasse).
- Uso da palavra-chave extends.
- Modificador de acesso protected e o uso de super() para reaproveitamento de construtores e métodos.
- Sobrescrita de métodos com a anotação @Override.

Unidade 6: Manipulação Dinâmica de Dados com Coleções (ArrayList)

Foco: Armazenar e gerenciar múltiplos objetos em memória de forma dinâmica utilizando a API de Coleções do Java.

Conteúdo:

- Limitações dos arrays estáticos tradicionais.
- A interface List e a classe ArrayList<T>.
- Operações básicas em listas: adicionar (add), buscar por índice (get), remover (remove), verificar tamanho (size).
- Iteração sobre listas utilizando o laço for-each.

Unidade 7: Persistência de Dados NoSQL e Operações CRUD em Java

Foco: Conectar a aplicação Java orientada a objetos a um banco de dados NoSQL (MongoDB) e executar as operações de persistência (CRUD).

Conteúdo:

- Conceitos de bancos de dados NoSQL orientados a documentos (JSON/BSON) vs. Bancos Relacionais.
- Introdução ao MongoDB: Databases, Collections e Documents.
- Integração do Java com MongoDB via MongoDB Java Driver.
- Implementação completa das operações de CRUD:
 - C (Create): Inserção de objetos convertidos em documentos BSON (insertOne).
 - R (Read): Consultas e buscas filtradas por atributos (find, Filters.eq).
 - U (Update): Atualização de dados existentes (updateOne, Updates.set).
 - D (Delete): Remoção de documentos (deleteOne).
- Organização do código com o padrão de projeto DAO (Data Access Object).

Unidade 1: Introdução ao Paradigma de POO e Configuração do Ambiente Java

1. A História do Java, Versão Atual e sua Importância na Computação

1.1. A Origem e a Filosofia "Write Once, Run Anywhere"

A história da linguagem Java começou no início dos anos 1990 dentro da empresa Sun Microsystems. Liderada por James Gosling (reconhecido como o "pai do Java"), uma equipe de engenheiros conhecida como Green Team tinha como meta inicial criar uma linguagem voltada para dispositivos eletrônicos inteligentes (como receptores de TV a cabo e eletrodomésticos).

Naquela época, a linguagem C++ era a dominante, mas apresentava sérios desafios: gerenciamento manual de memória complexo (ponteiros) e forte dependência do sistema operacional para o qual o programa era compilado. A equipe de Gosling projetou uma linguagem mais simples, segura, imune a ponteiros e com uma ideia revolucionária: o código escrito não seria compilado diretamente para o código de máquina do processador, mas sim para um formato intermediário chamado **Bytecode**.

Esse Bytecode é interpretado por uma máquina virtual chamada JVM (Java Virtual Machine). Com isso, nasceu o lema histórico do Java:

"Write Once, Run Anywhere" (Escreva uma vez, execute em qualquer lugar).

Lançada oficialmente em 1995, a linguagem explodiu em popularidade junto com o crescimento da internet, pois permitia rodar pequenos aplicativos interativos direto nos navegadores web (os Applets).

1.2. Evolução, Aquisição pela Oracle e a Versão Atual

Em 2010, a Oracle Corporation adquiriu a Sun Microsystems, assumindo a evolução da linguagem. A partir do Java 10 (2018), a Oracle adotou um ritmo de lançamentos semestral (uma nova versão a cada 6 meses) e uma versão com Suporte de Longo Prazo (LTS - Long Term Support) a cada 2 anos.

Versão de Referência LTS para Mercado e Ensino: Java 21 LTS (lançada no final de 2023, trazendo suporte estendido, Threads Virtuais e melhorias de desempenho).

Cadência Semestral: O ecossistema continua evoluindo rapidamente com as versões Java 22 e 23 (2024), mantendo compatibilidade total com o código legado.

1.3. Por que o Java é Importante na Ciência da Computação?

Para um estudante de Ciência da Computação, dominar o Java abre portas em diversas frentes da tecnologia:

- **Sistemas Corporativos e Bancários:** Bancos, plataformas de e-commerce e grandes multinacionais usam Java no seu ecossistema principal (backend) devido à sua alta performance, robustez e segurança.
- **Desenvolvimento Mobile (Android):** As bases do sistema operacional Android e grande parte das suas APIs foram construídas sobre a sintaxe do Java.
- **Big Data e Engenharia de Dados:** Frameworks famosos de processamento de dados massivos, como Apache Hadoop e Apache Spark, rodeiam o ambiente da JVM.
- **Base para Aprendizado:** Sendo uma linguagem fortemente tipada e puramente orientada a objetos, o Java obriga o aluno a entender conceitos cruciais de arquitetura de software, organização de memória e código limpo (Clean Code).

2. O Funcionamento Interno: Entendendo JVM, JRE e JDK

Para não confundir as siglas no dia a dia da faculdade, guarde esta relação:

Sigla	Nome Completo	O que é e para que serve?
JVM	Java Virtual Machine	A Máquina Virtual que lê e executa o Bytecode (.class) no sistema operacional.
JRE	Java Runtime Environment	O ambiente necessário para executar sistemas Java. Contém a JVM + bibliotecas padrão.
JDK	Java Development Kit	O Kit completo para desenvolvedores. Contém a JRE + Compilador (javac) e ferramentas. É este que instalaremos!

Fluxo de Execução:

Seu Código Fonte (.java) → Compilador (javac) → Bytecode (.class) →
Máquina Virtual (JVM) → Execução na tela.

3. Introdução à Programação Orientada a Objetos (POO)

3.1. Programação Estruturada vs. Orientada a Objetos

Até agora, é comum programarmos pensando em funções/procedimentos isolados e variáveis soltas (Programação Estruturada). Em sistemas reais e complexos, essa abordagem dificulta a manutenção do software.

A Programação Orientada a Objetos (POO) surge para organizar o código agrupando dados e comportamentos em uma única estrutura chamada Objeto.

3.2. Analogia do Mundo Real

Imagine um Carro da vida real:

- **Características / Dados (Atributos):** Cor, Marca, Modelo, Velocidade Atual.
- **Ações / Comportamentos (Métodos):** Acelerar(), Frear(), LigarFarol(), Buzinar().

Na POO, nós trazemos esse modelo mental para dentro do código!

4. Configuração do Ambiente no Laboratório: VS Code + Java

Em nossas aulas de laboratório na FMU, utilizaremos o Visual Studio Code (VS Code). Siga o passo a passo para deixar o seu computador pessoal igual ao do laboratório:

Passo 1: Instalação do JDK (Java Development Kit)

1. Baixe o instalador do JDK 17 LTS ou JDK 21 LTS (recomendamos distribuições gratuitas como Adoptium / Eclipse Temurin ou Oracle JDK).
2. Execute o instalador mantendo os caminhos padrão.
3. Para testar se a instalação funcionou, abra o Prompt de Comando (Terminal) e digite:

```
Bash
```

```
java -version
```

(Se aparecer a versão do Java no texto, deu tudo certo!)

Passo 2: Instalação do VS Code

1. Acesse code.visualstudio.com e faça o download do VS Code para o seu sistema operacional.
2. Instale o programa avançando pelas telas de configuração padrão.

Passo 3: Extensões Obrigatórias para Java no VS Code

O VS Code é um editor leve que ganha "superpoderes" através de extensões. Para programar em Java, instalaremos o pacote oficial mantido pela Microsoft e Red Hat:

1. Abra o VS Code e clique no ícone de Extensões no menu lateral esquerdo (ou pressione Ctrl + Shift + X).
2. Pesquise por: Extension Pack for Java (da Microsoft) e clique em Install.

Ao instalar este pacote principal, ele automaticamente instalará as 6 extensões essenciais:

- **Language Support for Java™ by Red Hat:** Traz o autocompletar inteligente, detecção de erros em tempo real e navegação nas classes.
- **Debugger for Java:** Permite executar o código linha por linha (depuração) para encontrar erros de lógica.
- **Project Manager for Java:** Organiza a estrutura de pastas e arquivos do seu projeto Java no painel do VS Code.
- **Test Runner for Java:** Executa testes unitários automáticos.
- **Maven for Java:** Gerenciador de bibliotecas e dependências externas.
- **IntelliCode:** Autocompletar avançado baseado em IA para sugerir o que digitar em seguida.

5. Nosso Primeiro Exemplo Prático: "Olá, Mundo!"

Abra o VS Code, crie um arquivo chamado PrimeiroPrograma.java e digite o código abaixo:

```
// Em Java, TODO o código fica obrigatoriamente dentro de uma Classe
public class PrimeiroPrograma{

    // O método 'main' é o ponto de partida onde a execução do programa começa
    public static void main(String[] args) {

        // Exibe o texto entre aspas no console do usuário
        System.out.println("Olá, turma de Ciência da Computação da FMU!");

    }
}
```

Entendendo a Estrutura Básica:

- **public class PrimeiroPrograma:** O nome da classe principal deve ser exatamente igual ao nome do arquivo salvo no computador (PrimeiroPrograma.java).
- **public static void main(String[] args):** É a porta de entrada. Quando você manda o Java rodar o programa, a JVM procura exatamente essa assinatura de método.
- **System.out.println(...):** Comando padrão do Java para exibir informações na tela e pular uma linha.

Unidade 2: Fundamentos do Java — Entrada, Processamento e Saída de Dados

A documentação oficial do JAVA pode ser acessada em: <https://dev.java/learn/>

1. A Anatomia de um Programa em Java

Na Unidade 1, preparamos nosso ambiente no VS Code e executamos nosso primeiro programa em Java. Agora, vamos entender como o computador processa as instruções no código.

Todo software, do mais simples ao mais complexo, opera baseado em três etapas fundamentais:

```
[ Entrada de Dados ] ---> [ Processamento na RAM ] ---> [ Saída de Dados ]  
(Teclado / Scanner)      (Cálculos e Operações)      (Console / Tela)
```

1. **Entrada de Dados:** O programa recebe informações do mundo exterior (dados digitados pelo usuário no teclado).
2. **Processamento:** O programa armazena as informações na memória, realiza cálculos, comparações e manipulações.
3. **Saída de Dados:** O programa exibe os resultados formatados na tela do console.

2. Variáveis e Tipos de Dados em Java

2.1. O que é uma Variável?

Pense em uma variável como uma caixinha etiquetada localizada na memória RAM do computador. Cada caixinha possui três características principais:

- **Nome (Identificador):** A etiqueta para podermos referenciar a caixa no código (ex: idade, nota1, nomeAluno).
- **Tipo:** Define o formato e a quantidade de memória que a caixa pode guardar (números inteiros, decimais, textos).
- **Valor:** O conteúdo guardado dentro da caixa naquele momento.

Convenção camelCase e Sensibilidade de Caixa:

- Em Java, nomes de variáveis devem começar com letra minúscula.

- Se o nome for composto por mais de uma palavra, usamos a convenção camelCase (ex: mediaFinal, dataNascimento, raAluno).
- O Java é Case Sensitive: a variável nota é completamente diferente de Nota.

2.2. Tipos Primitivos de Dados

O Java é uma linguagem fortemente tipada. Isso significa que toda variável precisa ter seu tipo declarado antes de ser utilizada e o compilador verifica rigidamente o cumprimento dessas regras para garantir a segurança da memória e a prevenção de erros em tempo de execução.

Diferente dos objetos (que são alocados na memória Heap e contêm métodos e atributos complexos), os tipos primitivos são os blocos fundamentais e indivisíveis da linguagem. Eles armazenam seus valores brutos diretamente na memória Stack, garantindo altíssima velocidade de leitura, escrita e baixo consumo de memória.

A linguagem Java possui exatamente 8 tipos primitivos, divididos em 4 grandes categorias:

1. Inteiros (Números sem casas decimais)

Representam valores numéricos inteiros (positivos, negativos ou zero) sinalizados no padrão de complemento de dois.

byte (8 bits / 1 byte):

- **Intervalo de valores:** De -128 até 127 (-2^7 a 2^7-1).
- **Valor Padrão** (Atributo de Classe): 0
- **Uso Recomendado:** Economia extrema de memória em grandes coleções/arrays de dados brutos ou leitura de dados binários vindo de arquivos, redes e fluxos de I/O (streams).
- **Exemplo:**

```
byte nivelSinal = 85;  
byte idade = 21;
```

short (16 bits / 2 bytes):

- **Intervalo de valores:** De -32.768 até 32.767 (-2^{15} a $2^{15}-1$).

- **Valor Padrão:** 0
- **Uso Recomendado:** Processamento de sistemas embarcados de baixo consumo de memória ou manipulação de buffers de áudio/vídeo onde o int seria excessivo.
- **Exemplo:**

```
short anoNascimento = 2004;  
short quantidadeEstoque = 15000;
```

int (32 bits / 4 bytes):

- **Intervalo de valores:** De -2.147.483.648 até 2.147.483.647 (-2^{31} a $2^{31}-1$).
- **Valor Padrão:** 0
- **Uso Recomendado:** O tipo padrão da linguagem Java para a maioria das contagens, índices de laços de repetição (for), quantidades e identificadores genéricos. É o tipo assumido por padrão para qualquer literal inteiro digitado no código.
- **Exemplo:**

```
int raAluno = 20261015;  
int contador = 0;
```

long (64 bits / 8 bytes):

- **Intervalo de valores:** De -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807 (-2^{63} a $2^{63}-1$).
- **Valor Padrão:** 0L
- **Uso Recomendado:** Utilizado para representar contadores gigantescos, carimbos de data/hora em milissegundos (timestamps do sistema), distâncias astronômicas e IDs primários em bancos de dados corporativos.
- **Sufixo Obrigatório:** Literais long que ultrapassam o limite de um int exigem o sufixo L ou l ao final do número (recomenda-se o L maiúsculo para evitar confusão com o dígito 1).
- **Exemplo:**

```
long populacaoMundial = 8100000000L;  
long timestampAtual = System.currentTimeMillis();
```

2. Ponto Flutuante / Decimais (Números com casas decimais)

Representam números reais (positivos ou negativos) com frações, baseados na especificação internacional IEEE 754.

float (32 bits / 4 bytes — Precisão Simples):

- **Precisão:** Aproximadamente 6 a 7 dígitos decimais significativos.
- **Intervalo aproximado:** $\pm 1,4 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$.
- **Valor Padrão:** 0.0f
- **Uso Recomendado:** Usado em cenários de renderização gráfica 3D, física de jogos e bibliotecas que necessitam economizar memória RAM ao guardar matrizes gigantescas de números reais onde a precisão extrema não seja crítica.
- **Sufixo Obrigatório:** Exige o sufixo f ou F ao final do número, pois o Java interpreta qualquer literal decimal como double por padrão.
- **Exemplo:**

```
float temperaturaSombra = 26.5f;
```

```
float taxaJuros = 1.25F;
```

double (64 bits / 8 bytes — Precisão Dupla):

- **Precisão:** Aproximadamente 15 a 17 dígitos decimais significativos.
- **Intervalo aproximado:** $\pm 4,9 \times 10^{-324}$ a $\pm 1,8 \times 10^{308}$.
- **Valor Padrão:** 0.0d (ou simplesmente 0.0)
- **Uso Recomendado:** O tipo padrão do Java para qualquer cálculo matemático real, estatísticas, cálculos científicos e geometrias.
- **Sufixo Opcional:** Pode utilizar o sufixo d ou D, porém não é necessário.
- **Exemplo:**

```
double notaN1 = 8.75;
```

```
double pi = 3.141592653589793;
```

⚠ **Atenção Financeira (Erro Clássico de Ponto Flutuante):**

Tipos **float** e **double** **NÃO** devem ser utilizados para **cálculos financeiros** (como moedas, saldos bancários ou impostos), pois a representação binária do padrão IEEE 754 causa pequenas **imprecisões de arredondamento em dízimas**. Para valores monetários exatos, a biblioteca padrão do Java fornece a classe **java.math.BigDecimal**.

3. Caractere

char (16 bits / 2 bytes — Sem Sinal):

- **Intervalo de valores:** De 0 até 65.535 (em hexadecimal: '\u0000' a '\uffff').
- **Valor Padrão:** '\u0000' (caractere nulo / null character).
- **O que armazena?** Armazena um único caractere mapeado na tabela internacional Unicode (o que permite representar letras tradicionais, símbolos matemáticos, caracteres acentuados do português e até ideogramas asiáticos).
- **Delimitação obrigatória:** Deve ser declarado utilizando aspas simples ' ' (aspas duplas " " são exclusivas para objetos String).
- **Comportamento Numérico:** Como o char armazena o código numérico da tabela Unicode, é possível realizar operações aritméticas diretamente com ele.
- **Exemplo:**

```
char genero = 'M';  
char conceitoFinal = 'A';  
char simboloAlfa = '\u03B1'; // Caractere grego α  
// Avançando um caractere na tabela Unicode ('A' + 1 vira 'B')  
char proximaLetra = (char) (conceitoFinal + 1);
```

4. Lógico / Booleano

boolean (Representação dependente da JVM, tipicamente 1 byte/8 bits na memória):

- **Valores Possíveis:** Apenas dois valores literais reservados: true (verdadeiro) ou false (falso).
- **Valor Padrão:** false
- **Uso Recomendado:** Controle de fluxo de execução, flags de estado de objetos, validação de expressões condicionais e estruturas de repetição.

➤ Exemplo:

```
boolean alunoAprovado = true;
boolean possuiMatriculaAtiva = false; boolean eMaiorDeIdade = (idade >= 18);
```

Tabela Resumo dos Tipos Primitivos

Categoria	Tipo	Tamanho (bits/bytes)	Valor Mínimo	Valor Máximo	Valor Padrão	Exemplo
Inteiro	byte	8 bits (1B)	-128	127	0	byte b = 100;
Inteiro	short	16 bits (2B)	-32.768	32.767	0	short s = 20000;
Inteiro	int	32 bits (4B)	-2.147.483.648	2.147.483.647	0	int i = 500000;
Inteiro	long	64 bits (8B)	-2 ⁶³	2 ⁶³ -1	0L	long l = 9000000L;
Ponto Flutuante	float	32 bits (4B)	≈±1,4×10 ⁻⁴⁵	≈±3,4×10 ³⁸	0.0f	float f = 7.5f;
Ponto Flutuante	double	64 bits (8B)	≈±4,9×10 ⁻³²⁴	≈±1,8×10 ³⁰⁸	0.0d	double d = 89.90;
Caractere	char	16 bits (2B)	0 ('\u0000')	65.535 ('\uffff')	\u0000'	char c = 'X';
Booleano	boolean	Depende da JVM	FALSE	TRUE	FALSE	boolean active = true;

2.3. O Tipo de Referência: String

Para armazenar textos e frases inteiras, utilizamos a classe **String**. Como **String** é uma classe (e não um tipo primitivo), seu nome começa com letra maiúscula e o texto deve estar entre aspas duplas .

```
String nome = "Robson Carmo";
String curso = "Ciência da Computação";
```

3. Operadores Aritméticos e Conversão de Tipos (Casting)

3.1. Operadores Aritméticos

- + (Soma / Concatenação de textos)
- - (Subtração)
- * (Multiplicação)

- / (Divisão)
- % (Módulo — Retorna o resto da divisão inteira)

⚠ Atenção com a Divisão Inteira!

Em Java, a divisão entre dois números inteiros resulta em outro número inteiro (a parte fracionária é descartada!).

- Exemplo: 5 / 2 resulta em 2.
- Para obter 2.5, ao menos um dos números deve ser decimal: 5.0 / 2 ou (double) 5 / 2.

3.2. Conversão de Tipos (Casting)

O **Casting** ocorre quando convertemos o tipo de uma variável para outro:

- **Casting Implícito (Automático):** Ocorre de um tipo menor para um maior sem perda de dados.

```
int valorInt = 10;  
Double valorDouble = valorInt; // Automaticamente vira 10.
```

- **Casting Explícito (Forçado):** Necessário ao converter um tipo para outro. Devemos indicar o tipo desejado entre parênteses:

```
double preco = 99.99;  
int precoInteiro = (int) preco; // Vira 99 (descarta a fração)
```

4. Saída de Dados com System.out

Para exibir mensagens e resultados no console, utilizamos:

- **System.out.print():** Imprime a mensagem e mantém o cursor na mesma linha.
- **System.out.println():** Imprime a mensagem e pula para a linha de baixo.
- **System.out.printf():** Imprime a mensagem com formatação de casas decimais.

```
String aluno = "Maria";double media = 8.7561;
// Exemplo com concatenação (+)
System.out.println("O aluno " + aluno + " ficou com média " + media);
// Exemplo formatado com %.2f (arredonda para 2 casas decimais na exibição)
System.out.printf("O aluno %s ficou com média %.2f\n", aluno, media);
```

5. Entrada de Dados com a Classe Scanner

Para ler dados digitados pelo usuário via teclado, utilizamos a classe **Scanner** do pacote **java.util**.

Passo a Passo de Uso do Scanner:

1. Importar a classe: `import java.util.Scanner;` (no topo do arquivo).
2. Instanciar o leitor: `Scanner leitor = new Scanner(System.in);`
3. Capturar a entrada de acordo com o tipo:
 - `leitor.nextInt()` → lê um inteiro (int).
 - `leitor.nextDouble()` → lê um decimal (double).
 - `leitor.next()` → lê uma palavra simples.
 - `leitor.nextLine()` → lê uma linha/frase inteira.
4. Fechar o leitor ao terminar: `leitor.close();`

6. Exemplo Prático Completo Integrado

Programa completo unindo Entrada, Processamento e Saída:

```
import java.util.Scanner;

public class CalculadoraMedia {

    public static void main(String[] args) {
        // 1. Instanciando o leitor do teclado
        Scanner leitor = new Scanner(System.in);

        System.out.println("=== SISTEMA DE NOTAS - FMU ===");

        System.out.print("Digite o nome do aluno: ");
        String nome = leitor.nextLine();

        System.out.print("Digite a Nota N1: ");
        double n1 = leitor.nextDouble();

        System.out.print("Digite a Nota N2: ");
        double n2 = leitor.nextDouble();
```

```
// 2. Processamento: Média ponderada (N1 peso 40%, N2 peso 60%)[cite: 1]
double mediaFinal = (n1 * 0.4) + (n2 * 0.6);

// 3. Saída de Dados
System.out.println("\n--- RESULTADO FINAL ---");
System.out.println("Aluno: " + nome);
System.out.printf("Média Semestral: %.2f\n", mediaFinal);

// Encerrando o recurso
leitor.close();
}
```

Exercícios de Fixação

Questão 1

Considere o seguinte excerto sobre a arquitetura da linguagem Java:

"A portabilidade da linguagem Java deve-se ao fato de seus programas não serem compilados diretamente para o código de máquina do processador onde foram desenvolvidos."

A respeito da compilação e execução de sistemas Java, avalie as asserções a seguir e a relação proposta entre elas:

I. O código-fonte gravado em um arquivo de extensão .java é transformado pelo compilador javac em um formato intermediário independente de plataforma chamado Bytecode, salvo em arquivos de extensão .class.

PORQUE

II. A Máquina Virtual Java (JVM) existente em cada sistema operacional é responsável por interpretar o Bytecode e traduzi-lo para as instruções nativas do hardware específico.

A respeito dessas asserções, assinale a opção correta:

- (A) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- (B) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- (C) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- (D) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- (E) As asserções I e II são proposições falsas.

Questão 2

Um estudante de Ciência da Computação escreveu o seguinte trecho de código em Java para calcular a média de três avaliações inteiras:

```
int nota1 = 7;
int nota2 = 8;
int nota3 = 7;
double media = (nota1 + nota2 + nota3) / 3;
System.out.println(media);
```

Ao executar o programa, o valor impresso no console foi 7.0, embora a soma das notas resulte em 22 e a média real exata seja 7.3333....

Qual alternativa explica a causa desse comportamento e apresenta a correção adequada?

- (A) O tipo double não aceita operações com variáveis inteiras, devendo todas ser declaradas como String.
- (B) Como a soma entre parênteses e o divisor 3 são inteiros (int), o Java realiza uma divisão inteira truncando os decimais antes da atribuição. A correção é alterar o divisor para 3.0.
- (C) A variável media deveria ter sido declarada como float, pois o double arredonda números para baixo.
- (D) O compilador Java apresentou um erro de sintaxe e não deveria ter executado.
- (E) A classe System.out.println só imprime números inteiros por padrão.

Questão 3

Analisar o trecho de código Java a seguir:

```
double saldoConta = 1250.85;
int saqueSemCentavos = (int) saldoConta;
System.out.println(saqueSemCentavos);
```

Assinale a alternativa que indica o valor que será exibido no console após a execução:

- (A) 1250.85
- (B) 1251

- (C) 1250
- (D) 0
- (E) Erro de compilação devido ao uso de casting.

Questão 4

Considere o programa Java abaixo:

```
import java.util.Scanner;
public class TesteLeitura {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt();
        int b = sc.nextInt();
        System.out.println(a + b + " result");
    }
}
```

Se o usuário fornecer os valores 5 e 10 como entrada, o que será exibido na tela?

- (A) "a + b result"
- (B) "510 result"
- (C) "15 result"
- (D) Erro de execução no método println.
- (E) "5 10 result"

Questão 5

Em projetos orientados a objetos em Java, a padronização de nomenclatura é vital para a legibilidade do código. Analise os identificadores de variáveis/métodos abaixo:

- I. salario_bruto
- II. valorTotalConta
- III. 1aParcela
- IV. calcularMediaGeral()

Assinale a opção que indica os identificadores válidos na sintaxe e que seguem a convenção camelCase exigida no Java:

- (A) I e III, apenas.
- (B) II e IV, apenas.
- (C) I, II e IV, apenas.
- (D) III e IV, apenas.
- (E) Todos são válidos e seguem a convenção.

Exercícios Práticos de Programação

Exercício Prático 1: Conversor de Temperatura (°F para °C)

Desafio: Escreva um programa em Java que solicite ao usuário uma temperatura em graus Fahrenheit (F), calcule e exiba a temperatura correspondente em graus Celsius (C).

Fórmula:

$$C = \frac{5}{9} \times (F - 32)$$

Exercício Prático 2: Calculadora de Salário Líquido

Desafio: Escreva um programa que leia o valor ganho por hora por um funcionário, a quantidade de horas trabalhadas no mês e a porcentagem de desconto do INSS. O programa deve calcular e exibir o Salário Bruto, o Valor Descontado e o Salário Líquido.

Exercício Prático 3: Decomposição de Tempo (Segundos para Horas, Minutos e Segundos)

Desafio: Escreva um programa em Java que receba do usuário um tempo total em segundos (número inteiro) e converta-o para o formato Horas : Minutos : Segundos, utilizando operadores de divisão (/) e resto da divisão (%).

Unidade 3: Lógica de Controle — Estruturas Condicionais e Laços de Repetição

1. Introdução ao Fluxo de Execução

Até a Unidade 2, nossos programas seguiam uma execução sequencial linear: o código começava na primeira linha do método main e executava instrução por instrução, de cima para baixo, até a última linha.

Na vida real, os sistemas de software precisam tomar decisões e repetir tarefas automaticamente:

- Se a média do aluno for maior ou igual a 6.0, ele está aprovado; caso contrário, está de exame.
- Enquanto o usuário não digitar uma opção válida, o sistema deve continuar exibindo o menu.

Nesta unidade, aprenderemos a controlar o fluxo do programa em Java utilizando **Estruturas de Decisão (condicionais)** e **Laços de Repetição (loops)**.

2. Operadores Relacionais e Lógicos

Para que o programa consiga "tomar decisões", precisamos realizar comparações. O resultado de qualquer comparação em Java é sempre um valor booleano (true ou false).

2.1. Operadores Relacionais

Utilizados para comparar dois valores:

Operador	Significado	Exemplo	Resultado (a = 10, b = 5)
==	Igual a	a == b	FALSE
!=	Diferente de	a != b	TRUE
>	Maior que	a > b	TRUE
<	Menor que	a < b	FALSE
>=	Maior ou igual a	a >= 10	TRUE
<=	Menor ou igual a	b <= 5	TRUE

⚠ **Atenção:** O operador = é de atribuição (guardar valor na variável). O operador == é de comparação (verificar se são iguais).

2.2. Operadores Lógicos

Utilizados para combinar duas ou mais condições:

- **&& (E lógico / AND):** Retorna true apenas se todas as condições forem verdadeiras.
- **|| (OU lógico / OR):** Retorna true se pelo menos uma das condições for verdadeira.
- **! (NÃO lógico / NOT):** Inverte o valor booleano (se for true vira false, e vice-versa).

```
int idade = 20;

boolean possuiCarteira = true;

// Ambas as condições precisam ser true
boolean podeDirigir = (idade >= 18) && (possuiCarteira == true); // true
```

3. Estruturas Condicionais

3.1. Estrutura if / else if / else

A estrutura if (SE) avalia uma condição booleana. Se for verdadeira, executa o bloco de código dentro das chaves {}. Caso contrário, o código do bloco else (SENÃO) é acionado.

```
import java.util.Scanner;

public class VerificadorNota {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);

        System.out.print("Digite a média do aluno: ");
        double media = leitor.nextDouble();

        // Estrutura Condicional Encadeada
        if (media >= 6.0) {
            System.out.println("Aluno Aprovado!");
        } else if (media >= 4.0) {
            System.out.println("Aluno em Exame de Recuperação.");
        } else {
            System.out.println("Aluno Reprovado.");
        }

        leitor.close();
    }
}
```

3.2. Operador Ternário (Condicional Sucinta)

Para atribuições condicionais simples em uma única linha, podemos usar o operador ternário (condição ? valor_se_true : valor_se_false):

```
double media = 7.5;
String resultado = (media >= 6.0) ? "Aprovado" : "Reprovado";
System.out.println("Situação: " + resultado);
```

3.3. Seleção Múltipla com switch-case

Quando precisamos testar uma única variável contra múltiplos valores exatos (como as opções de um menu), o switch-case torna o código mais limpo do que vários if-else encadeados.

```
import java.util.Scanner;

public class MenuOpcoes {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);

        System.out.println("=== MENU PRINCIPAL ===");
        System.out.println("1 - Cadastrar Aluno");
        System.out.println("2 - Listar Alunos");
        System.out.println("3 - Sair");
        System.out.print("Opção: ");
        int opcao = leitor.nextInt();

        switch (opcao) {
            case 1:
                System.out.println("Iniciando cadastro de aluno...");
                break; // O 'break' interrompe a execução do switch
            case 2:
                System.out.println("Exibindo lista de alunos...");
                break;
            case 3:
                System.out.println("Saindo do sistema...");
                break;
            default:
                System.out.println("Opção inválida!");
                break;
        }

        leitor.close();
    }
}
```

4. Estruturas de Repetição (Laços / Loops)

As estruturas de repetição permitem executar um mesmo bloco de código várias vezes até que uma determinada condição deixe de ser satisfeita.

4.1. Laço while (Enquanto)

Testa a condição antes de executar o bloco. É ideal quando não sabemos previamente quantas vezes o laço precisará repetir.

```
import java.util.Scanner;

public class ValidacaoEntrada {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        double nota = -1;

        // Repete enquanto a nota digitada for inválida (fora do intervalo 0 a 10)
        while (nota < 0 || nota > 10) {
            System.out.print("Digite uma nota válida (0 a 10): ");
            nota = leitor.nextDouble();
        }

        System.out.println("Nota cadastrada com sucesso: " + nota);
        leitor.close();
    }
}
```

4.2. Laço do-while (Faça... Enquanto)

Executa o bloco de código pelo menos uma vez antes de testar a condição no final. Muito utilizado na construção de menus interativos.

```
import java.util.Scanner;

public class MenuComDowhile {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        int opcao;

        do {
            System.out.println("\n=== MENU ===");
            System.out.println("1 - Novo Cálculo");
            System.out.println("0 - Sair");
            System.out.print("Escolha: ");
            opcao = leitor.nextInt();
        } while (opcao != 0); // Repete se a opção for diferente de 0

        System.out.println("Programa encerrado.");
        leitor.close();
    }
}
```

4.3. Laço for (Para)

É a estrutura de repetição ideal quando sabemos exatamente quantas vezes o bloco deve ser executado. O for agrupa três elementos em sua sintaxe:

1. **Inicialização:** Cria e inicializa a variável contadora.
2. **Condição:** Condição de parada verificada antes de cada volta.
3. **Incremento:** Atualiza o contador ao final de cada volta.

```
public class TabuadaFor {  
    public static void main(String[] args) {  
        int numero = 5;  
  
        System.out.println("=== TABUADA DO " + numero + " ===");  
  
        // Contagem de 1 até 10  
        for (int i = 1; i <= 10; i++) {  
            int resultado = numero * i;  
            System.out.printf("%d x %02d = %d\n", numero, i, resultado);  
        }  
    }  
}
```

5. Exercícios de Fixação

Questão 1

Considere o seguinte trecho de código escrito em linguagem Java:

```
int x = 10;  
int y = 0;  
  
while (x > 0) {  
    if (x % 2 == 0) {  
        y += x;  
    }  
    x -= 3;  
}  
  
System.out.println("y = " + y);
```

Após a execução completa do trecho de código acima, qual será o valor impresso para a variável y?

- (A) 30
- (B) 14
- (C) 10
- (D) 18
- (E) 0

Questão 2

Em Java, a avaliação de expressões lógicas utilizando os operadores && e || ocorre na modalidade de curto-circuito (short-circuit). Considere a expressão:

```
boolean resultado = (a > b) && (c++ > 5);
```

Se o valor inicial das variáveis for a = 2, b = 5 e c = 10, qual será o valor final das variáveis c e resultado após a execução da linha?

- (A) c = 11 e resultado = true
- (B) c = 11 e resultado = false
- (C) c = 10 e resultado = false
- (D) c = 10 e resultado = true
- (E) Ocorre um erro de compilação por tentar alterar a variável dentro da condição.

Questão 3

Analise o programa Java abaixo:

```
public class TesteSwitch {  
    public static void main(String[] args) {  
        int valor = 2;  
        int resultado = 0;  
  
        switch (valor) {  
            case 1:  
                resultado += 10;  
            case 2:  
                resultado += 20;  
            case 3:  
                resultado += 30;  
                break;  
            default:  
                resultado += 100;  
        }  
        System.out.println(resultado);  
    }  
}
```

Qual será o valor impresso no console?

- (A) 20
- (B) 60
- (C) 150
- (D) 50
- (E) 30

Questão 4

Qual dos seguintes blocos de código utilizando while é estritamente equivalente ao laço for (int i = 0; i < 5; i++) { System.out.print(i); }?

- (A) int i = 0; while (i < 5) { i++; System.out.print(i); }
- (B) int i = 0; while (i < 5) { System.out.print(i); i++; }
- (C) int i = 0; while (i <= 5) { System.out.print(i); i++; }
- (D) int i = 1; while (i < 5) { System.out.print(i); i++; }
- (E) int i = 5; while (i > 0) { System.out.print(i); i--; }

Questão 5

Considere uma regra de negócio para concessão de crédito universitário:

"O aluno terá direito ao crédito se a sua média semestral for maior ou igual a 7.0 OU se a renda familiar por pessoa for menor que 2 salários mínimos, DESDE QUE ele não tenha nenhuma reprovação por falta."

Sendo as variáveis booleanas:

- m: Média ≥ 7.0
- r: Renda por pessoa < 2 salários
- f: Possui reprovação por falta

A expressão booleana que representa corretamente essa regra em Java é:

- (A) (m && r) || f
- (B) m || (r && !f)
- (C) (m || r) && !f
- (D) !(m || r) && f
- (E) m && r && f

Exercícios Práticos de Programação

Exercício Prático 1: Classificador de IMC (Índice de Massa Corporal)

Desafio: Escreva um programa em Java que leia o peso (em kg) e a altura (em metros) de uma pessoa, calcule o IMC ($IMC = \frac{\text{peso}}{\text{altura}^2}$) e exiba a classificação do indivíduo de acordo com a tabela abaixo:

- $IMC < 18.5$: Abaixo do peso
- $18.5 \leq IMC < 25.0$: Peso normal
- $25.0 \leq IMC < 30.0$: Sobrepeso
- $IMC \geq 30.0$: Obesidade

Exercício Prático 2: Análise de Números (Maior, Menor e Média)

Desafio: Escreva um programa que leia 5 números inteiros fornecidos pelo usuário via laço for. Ao final, o programa deve exibir o Maior número digitado, o Menor número digitado e a Média Aritmética dos 5 números.

Exercício Prático 3: Validação de Senha com Tentativas Limitadas

Desafio: Escreva um programa que simule o bloqueio de um caixa eletrônico. O usuário possui uma senha cadastrada (ex: 1234). O programa deve dar até 3 tentativas para o usuário acertar a senha. Se o usuário acertar, exibe a mensagem "Acesso Permitido" e encerra. Se errar as 3 tentativas, exibe a mensagem "Cartão Bloqueado!".

Unidade 4: O Coração da POO — Classes, Objetos, Atributos, Métodos e Encapsulamento

1. Do Pensamento Estruturado para o Paradigma Orientado a Objetos

Nas unidades anteriores, nossos programas eram compostos por variáveis soltas na memória e funções sequenciais no main. Conforme os sistemas crescem, essa abordagem torna o código difícil de manter, inseguro e propenso a falhas.

A Programação Orientada a Objetos (POO) surge como uma filosofia de software que aproxima a estrutura do código da forma como enxergamos o mundo real. Em vez de tratar dados e instruções separadamente, nós os agrupamos em entidades autônomas chamadas Objetos.

PARADIGMA ESTRUTURADO	PARADIGMA ORIENTADO A OBJETOS
-----	-----
Variáveis e Funções soltas no código principal.	[OBJETO] ├ Atributos (Dados/Estado) └ Métodos (Comportamentos/Ações)

2. A Relação Fundamental: Classe vs. Objeto

Para entender a base da POO, é essencial dominar a diferença entre Classe e Objeto:

- **Classe (A Planta Baixa / O Molde):** É a especificação técnica ou abstração. Ela não ocupa espaço dinâmico na memória para armazenar dados do usuário; ela apenas define quais atributos e quais métodos os objetos criados a partir dela terão.
- **Objeto (A Instância Real):** É a concretização do molde na memória RAM do computador, criada através do operador new. Cada objeto possui seus próprios valores para os atributos definidos na classe.

Analogia Didática:

- **Classe:** A planta arquitetônica de uma casa.
- **Objeto:** As casas reais construídas na rua com base nessa mesma planta (uma pode ser azul com 2 quartos, outra amarela com 3 quartos).

3. Elementos de uma Classe: Atributos, Métodos e Construtores

3.1. Atributos (Estado)

Representam as características e informações que a classe armazena.

3.2. Métodos (Comportamento)

Representam as ações e operações que a classe é capaz de realizar. Métodos podem receber parâmetros e retornar valores (ou retornar void caso não devolvam nada).

3.3. Construtores

São métodos especiais chamados automaticamente no momento em que um objeto é instanciado com o operador new. Sua função principal é inicializar o estado inicial do objeto.

- Possuem exatamente o mesmo nome da classe.
- Não possuem tipo de retorno (nem mesmo void).

```
public class Aluno {  
    // 1. Atributos  
    String nome;  
    String ra;  
    double nota1;  
    double nota2;  
  
    // 2. Construtor Personalizado  
    public Aluno(String nome, String ra) {  
        this.nome = nome; // 'this' diferencia o atributo da classe do parâmetro recebido  
        this.ra = ra;  
    }  
  
    // 3. Método para calcular a média  
    public double calcularMedia() {  
        return (this.nota1 + this.nota2) / 2.0;  
    }  
}
```

4. O Pilar do Encapsulamento e Modificadores de Acesso

O Encapsulamento é o pilar da POO que protege os dados internos de um objeto contra alterações indevidas ou acessos diretos desautorizados de fora da classe. Ele funciona como a carcaça de uma cápsula de remédio, ocultando os detalhes internos e expondo apenas uma interface segura de interação.

4.1. Modificadores de Acesso

Em Java, controlamos o nível de visibilidade usando palavras-chave:

Modificador	Visibilidade	Onde pode ser acessado?
private	Restrita	Apenas dentro da própria classe onde foi declarado.
public	Livre	Por qualquer outra classe do projeto.
protected	Hereditária	Dentro da mesma classe, mesmo pacote ou por subclasses (veremos na Unidade 5).
(default)	Pacote	Apenas por classes dentro do mesmo pacote (quando não se digita nada).

4.2. Métodos Getters e Setters

Como os atributos devem ser mantidos como private, utilizamos métodos públicos para ler e alterar seus valores de forma controlada:

- **Getters (get):** Métodos de leitura que retornam o valor atual do atributo.
- **Setters (set):** Métodos de escrita que recebem um valor, aplicam validações de segurança e atualizam o atributo.

```
public class ContaBancaria {
    // Atributo privado encapsulado
    private double saldo;

    // Getter: permite consultar o saldo de forma segura
    public double getSaldo() {
        return this.saldo;
    }

    // Setter: valida o valor antes de alterar o saldo
    public void depositar(double valor) {
        if (valor > 0) {
            this.saldo += valor;
            System.out.println("Depósito de R$ " + valor + " realizado com sucesso.");
        } else {
            System.out.println("Erro: Valor de depósito deve ser positivo.");
        }
    }
}
```

5. Membros Estáticos (static)

A palavra-chave static indica que um atributo ou método pertence à Classe como um todo, e não a uma instância/objeto específico.

- **Atributo Estático:** Compartilhado entre todas as instâncias da classe na memória.

- **Método Estático:** Pode ser chamado diretamente pelo nome da Classe, sem a necessidade de criar um objeto com new (exemplo: Math.sqrt() ou System.out.println()).

6. Exemplo Completo Integrado: A Classe AlunoBCC

Abaixo está a implementação completa de uma classe encapsulada seguindo os padrões profissionais do Java:

```
public class AlunoBCC {
    // Atributos privados (Encapsulamento)
    private String nome;
    private String ra;
    private double notaN1;
    private double notaN2;

    // Atributo estático para contar quantos alunos foram criados no sistema
    private static int totalAlunosCadastrados = 0;

    // Construtor
    public AlunoBCC(String nome, String ra) {
        this.nome = nome;
        this.ra = ra;
        totalAlunosCadastrados++; // Incrementa o contador global da classe
    }

    // Métodos de Regra de Negócio
    public double calcularMediaSemestral() {
        return (this.notaN1 * 0.4) + (this.notaN2 * 0.6); // Média ponderada FMU
    }

    public boolean verificarAprovação() {
        return this.calcularMediaSemestral() >= 6.0;
    }

    // Getters e Setters com Validação
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getRa() { return ra; }

    public double getNotaN1() { return notaN1; }
    public void setNotaN1(double notaN1) {
        if (notaN1 >= 0.0 && notaN1 <= 10.0) {
            this.notaN1 = notaN1;
        } else {
            System.out.println("Erro: Nota N1 deve estar entre 0.0 e 10.0");
        }
    }

    public double getNotaN2() { return notaN2; }
    public void setNotaN2(double notaN2) {
        if (notaN2 >= 0.0 && notaN2 <= 10.0) {
            this.notaN2 = notaN2;
        } else {
            System.out.println("Erro: Nota N2 deve estar entre 0.0 e 10.0");
        }
    }

    // Método estático
    public static int getTotalAlunosCadastrados() {
        return totalAlunosCadastrados;
    }
}
```

Exercícios de Fixação

Questão 1

Um desenvolvedor júnior declarou todos os atributos de uma classe de sistema bancário como public. Em uma revisão de código (code review), o arquiteto de software solicitou a refatoração para aplicar o princípio de encapsulamento.

Qual alteração e justificativa técnica atendem corretamente à solicitação do arquiteto?

- (A) Alterar a visibilidade dos atributos para private e criar métodos get e set, garantindo que o acesso e a modificação do estado do objeto sejam controlados e validados pela própria classe.
- (B) Alterar os atributos para static, permitindo que o banco de dados os acesse sem instanciar a classe.
- (C) Manter os atributos como public, mas adicionar a palavra-chave final a todos para proibir que o valor mude.
- (D) Mudar os atributos para protected e remover todos os construtores da classe.
- (E) Criar um método main dentro da classe para ler as variáveis diretamente do teclado.

Questão 2

Considere o seguinte trecho de código Java:

```
public class Teste {  
    public static void main(String[] args) {  
        Aluno a1 = new Aluno("Carlos", "123");  
        Aluno a2 = a1;  
        a2.setNome("Roberto");  
  
        System.out.println(a1.getNome());  
    }  
}
```

O que será impresso no console ao executar o código acima?

- (A) Carlos
- (B) Roberto
- (C) null
- (D) Ocorre um erro de compilação por duplicidade de variáveis.
- (E) Ocorre uma exceção em tempo de execução (NullPointerException).

Questão 3

Analise as afirmações sobre a palavra-chave static em Java:

- I. Um método estático não pode acessar diretamente atributos de instância (não estáticos) da classe sem antes criar um objeto.
- II. Atributos estáticos possuem cópias individuais e exclusivas para cada objeto criado com o operador new.
- III. Métodos estáticos podem ser chamados diretamente através do nome da classe, sem a necessidade de instanciação.

É correto o que se afirma em:

- (A) I, apenas.
- (B) II, apenas.
- (C) I e III, apenas.
- (D) II e III, apenas.
- (E) I, II e III.

Questão 4

Sobre os construtores na linguagem Java, assinale a alternativa correta:

- (A) Se o programador não escrever nenhum construtor na classe, o compilador Java insere automaticamente um construtor padrão sem parâmetros.
- (B) Construtores devem ter o tipo de retorno void obrigatoriamente.
- (C) Uma classe pode ter no máximo um único construtor em seu código.
- (D) O construtor é executado toda vez que um método da classe é chamado pelo programa.
- (E) O uso da palavra-chave this dentro do construtor gera um erro de compilação.

Questão 5

No contexto de modelagem orientada a objetos, os atributos e os métodos de uma classe correspondem, respectivamente, a:

- (A) Algoritmos e Dados brutos.
- (B) Estado e Comportamento.

- (C) Interfaces e Módulos.
- (D) Compilação e Execução.
- (E) Atribuição e Herança.

Exercícios Práticos de Programação

Exercício Prático 1: Modelagem da Classe Produto

Desafio: Crie uma classe chamada Produto com os atributos privados nome (String), preco (double) e quantidadeEstoque (int). Implemente um construtor, os métodos getters/setters (com validação para não permitir preço nem quantidade negativos) e os métodos adicionarEstoque(int qtd) e vender(int qtd).

Exercício Prático 2: Sistema de Retângulo

Desafio: Crie uma classe Retangulo com atributos privados largura e altura. Implemente métodos para calcular a área ($\text{Area} = \text{largura} \times \text{altura}$), o perímetro ($\text{Perimetro} = 2 \times (\text{largura} + \text{altura})$) e um método que retorne se o retângulo é um quadrado ($\text{largura} == \text{altura}$).

Exercício Prático 3: Instanciação e Teste no main

Desafio: Escreva uma classe TestaProduto com o método main que crie dois objetos da classe Produto (desenvolvida no Exercício 1), realize operações de venda e reabastecimento de estoque, e exiba os resultados no console.

Exercício Prático 4: Modelagem de Sistema de Emissão de Faturas

Desafio: Crie uma aplicação em Java para simular a emissão de faturas comerciais de uma empresa. O sistema deve seguir rigorosamente os princípios da Programação Orientada a Objetos e o Encapsulamento de dados, contendo a seguinte estrutura de classes e relacionamentos:

Classe Cliente:

- **Atributos privados:** razaoSocial (String), cnpj (String), inscricaoEstadual (String), endereco (String), email (String) e celular (String).
- **Métodos:** Construtor parametrizado com validação dos campos obrigatórios (razaoSocial e cnpj), além de Getters e Setters.

Classe Produto:

- **Atributos privados:** codigo (int), nome (String), ncm (String) e valor (double).
- **Métodos:** Construtor parametrizado, Getters e Setters (com validação para impedir valor negativo ou zero).

Classe Fatura:

- **Atributos privados:** codigo (int), cliente (Cliente - associação/composição), dataEmissao (String), produtos (ArrayList de Produto) e valorTotal (double).
- **Métodos:**
 - Construtor recebendo codigo, cliente e dataEmissao.
 - adicionarProduto(Produto p): Método que adiciona um produto à lista interna e atualiza automaticamente o valorTotal da fatura.
 - removerProduto(Produto p): Método que remove um produto da lista e recalcula o valorTotal.
 - imprimirFatura(): Exibe no console um cabeçalho completo com os dados do cliente, a listagem dos produtos com seus valores/NCM e o valor final consolidado.

Classe TestaFatura (main):

- Instancie 1 cliente e pelo menos 3 produtos.
- Crie uma fatura associando o cliente criado.
- Adicione os produtos à fatura e acione o método de impressão do comprovante fiscal no console.

FMU
LEVEL UP

Bibliografia Básica

DEITEL, P. ;DEITEL, Harvey. Java: Como Programar. 10ª ed. São Paulo: Pearson, 2017. Disponível na Biblioteca Virtual.

<https://plataforma.bvirtual.com.br/Leitor/Publicacao/39590/pdf/0?code=EvX6a9r63l26HJWxyLCZjXQP0iJG4g1c9LA7whXv1g/mFLb4Li+HUSet4/qyXzOC04JsOHa9tEtPyPivdqKXOA==>

MANZANO, J. A. G.; COSTA JR., R. Programação de Computadores com Java. Érica, 2014. 127p. [Minha Biblioteca]

MANZANO, J. A. G. Programação de Computadores com C/C++. Érica, 06/2014. 120p. [Minha Biblioteca].

Bibliografia Complementar:

BARNES, D. J.; KOLLING, M. Programação Orientada a Objetos com Java: uma introdução prática usando o BlueJ - 4ª edição. São Paulo:

Pearson Prentice Hall, 2009. 480p. [Biblioteca Virtual Universitária].

MEILIR, P. Fundamentos do Desenho Orientado a Objeto com UML. São Paulo: Makron Books, 2001. 462p. [Biblioteca Virtual Universitária].

FÉLIX, R. Programação Orientada a Objetos. São Paulo: Pearson Education do Brasil, 2016. 164p. [Biblioteca Virtual Universitária].

KOFFMAN, E. B., WOLFGANG, P. T. Objetos, Abstração, Estrutura de Dados e Projeto Usando C++. Rio de Janeiro: LTC, 2008. 455p. [Minha Biblioteca].

DEITEL, P. ;DEITEL, Harvey. Java: Como Programar. 10ª ed. São Paulo: Pearson, 2017. Disponível na Biblioteca Virtual.

Periódicos Científicos:

Advances in Software Engineering - ISSN 1687-8655

Computación y Sistemas - ISSN 1405-5546

Computational & Applied Mathematics - ISSN 1807-0302

Journal of the Brazilian Computer Society - ISSN 0104-6500