DOMINANDO JAVA EM 1 ANO

Caderno de Exercícios

79 aulas de prática, do #00 ao #78

Java 25 LTS · 9 fases e 3 trilhas · 189 questões · 7 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

Como usar este caderno

Cada módulo corresponde a uma fase ou trilha do curso, com a lista das aulas que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas completos: escreva, compile e execute cada um.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Aulas	Questões
FASE 1	Primeiros passos na JVM	#00–#07	Q1–Q19
FASE 2	Objetos	#08–#15	Q20–Q38
FASE 3	A biblioteca padrão	#16–#22	Q39–Q58
FASE 4	Java funcional	#23–#27	Q59–Q73
FASE 5	Projetos e ferramentas	#28–#32	Q74–Q85
FASE 6	Por dentro da JVM	#33–#36	Q86–Q96
FASE 7	Concorrência	#37–#43	Q97–Q113
FASE 8	Qualidade e distribuição	#44–#46	Q114–Q121
FASE 9	O projeto final: o servidor HTTP Farol	#47–#52	Q122–Q134
TRILHA A	Web, API e bancos de dados: as Trilhas da Ilha	#53–#66	Q135–Q162
TRILHA B	Desktop e Android	#67–#70	Q163–Q170
TRILHA C	Jogos com libGDX	#71–#77	Q171–Q186
ENCERRAMENTO	O destino e o novo começo	#78	Q187–Q189
DESAFIOS	Projetos integradores que atravessam várias fases e trilhas	todas	D1–D7

Legenda: nas perguntas, `assim` indica código, nomes de classes e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

FASE 1

Primeiros passos na JVM

Artigos 00 a 07

#00 O Mapa da Jornada — o plano do curso
#01 Uma Linguagem para Torradeiras que Conquistou o Mundo — a História do Java
#02 Do Código ao Bytecode — JDK, javac, JVM e o Primeiro Programa
#03 Cada Valor em sua Caixa — Tipos Primitivos, Variáveis e Conversões
#04 Operadores, Texto e Conversa com o Usuário — Expressões, Strings e Entrada
#05 O Programa que Escolhe e Repete — if, switch e Laços
#06 Dividir para Resolver — Métodos, Parâmetros e Recursão
#07 Muitos Valores sob um Nome — Arrays

Perguntas conceituais

Q1 Onde, quando e por quem o Java foi criado? Qual era o objetivo original do projeto, e por que o fracasso desse objetivo acabou sendo a sorte da linguagem?

Q2 Explique a diferença entre **JDK**, **JRE** e **JVM**. Qual deles você instala para programar, e por que hoje quase não se fala mais em "instalar o JRE"?

Q3 O que é **bytecode**? Por que ele permite que o mesmo `.class` rode em Windows, Linux e macOS? Que ferramenta do JDK mostra o bytecode de uma classe?

Q4 Qual é a diferença entre rodar um programa com `javac Ola.java` seguido de `java Ola`, e rodar diretamente com `java Ola.java`? Em que situação cada forma é a mais adequada?

Q5 O que significa dizer que um programa Java "aquece"? Relacione a resposta com o interpretador e o compilador JIT.

Q6 Liste os oito tipos primitivos do Java com o tamanho de cada um. Qual deles não tem um tamanho fixo definido pela especificação da linguagem?

Q7 Em que situações a palavra `var` pode ser usada? Por que `var x;` não compila, e qual é o tipo inferido em `var lista = new ArrayList<>();`?

Q8 Por que comparar duas `String`s com `==` é um erro, mesmo quando às vezes "funciona"? Dê um exemplo em que `==` devolve `false` para textos iguais.

Exercícios de código

Q9 Sem executar, diga o que o programa imprime. Depois execute e confira.

```
void main() {
    int grande = Integer.MAX_VALUE;
    grande++;
    IO.println(grande);
    IO.println((byte) 200);
    IO.println(0.1 + 0.2);
    IO.println((int) 3.99);
    IO.println('a' + 1);
    IO.println((char) ('a' + 1));
    IO.println(7 / 2 + " " + 7 % 2 + " " + -7 / 2 + " " + -7 % 2);
}
```

Q10 Escreva um programa que conte quantos passos a sequência de Collatz leva para chegar a 1 a partir de 27. Regra: se `n` é par, vira `n / 2`; se é ímpar, vira `3n + 1`. Use um `long` e explique por quê.

Q11 Reescreva o `switch` abaixo como uma **switch expression** (com `->`), sem `break`, e garantindo que o compilador reclame se um valor ficar de fora.

```
enum Dia { SEG, TER, QUA, QUI, SEX, SAB, DOM }
```

```
String tipo;
switch (dia) {
    case SAB:
    case DOM:
        tipo = "fim de semana";
        break;
    default:
        tipo = "dia útil";
}
```

Q12 Use um `break` com **rótulo** para encontrar a primeira posição (linha, coluna) de um valor negativo numa matriz, saindo dos dois laços de uma vez.

Q13 O que o programa imprime? Explique com a ideia de que **o Java sempre passa parâmetros por valor**.

```
void trocar(int[] v) { v = new int[]{9, 9, 9}; }
void mexer(int[] v) { v[0] = 42; }
void dobrar(int x) { x = x * 2; }
```

```
void main() {
    int[] a = {1, 2, 3};
    int n = 5;
    trocar(a);
    mexer(a);
    dobrar(n);
    IO.println(java.util.Arrays.toString(a) + " " + n);
}
```

Q14 Qual versão de `f` é chamada em cada linha? Justifique pela ordem em que o compilador procura as sobrecargas.

```
void f(long x)     { IO.println("long"); }
void f(Integer x)  { IO.println("Integer"); }
void f(int... x)   { IO.println("varargs"); }
```

```
void main() {
    f(5);
    f(5L);
    f(Integer.valueOf(5));
    f();
}
```

Q15 Escreva a versão **recursiva** do máximo divisor comum pelo algoritmo de Euclides (`mdc(a, b) = mdc(b, a % b)`, com `mdc(a, 0) = a`). Depois explique o que causa um `StackOverflowError` numa recursão.

Q16 Escreva um método `void inverter(int[] v)` que inverta o array **no próprio lugar**, sem criar outro array. Mostre o resultado para `{1, 2, 3, 4, 5}` e para um array vazio.

Q17 Por que `IO.println(new int[]{1, 2})` imprime algo como `[I@1b6d3586` em vez dos números? Como imprimir um array corretamente? E como copiar um array de modo que alterar a cópia não altere o original?

Q18 Uma **matriz irregular** (*jagged*) tem linhas de tamanhos diferentes. Monte o triângulo de Pascal com 5 linhas usando `int[][]`, em que a linha `i` tem `i + 1` elementos, e imprima-o.

Q19 Escreva um programa que leia números inteiros digitados pelo usuário até ele digitar `fim`, e mostre a quantidade, a soma e a média. Trate a digitação de algo que não é número sem derrubar o programa.

FASE 2

Objetos

Artigos 08 a 15

#08 Moldes e Instâncias — Classes, Objetos e Construtores
#09 O que Fica Dentro, o que Fica Fora — Encapsulamento e Modificadores de Acesso
#10 Do Geral ao Específico — Herança e a Classe Object
#11 Contratos sem Implementação — Interfaces e Classes Abstratas
#12 Um Tipo, Várias Formas — Polimorfismo na Prática
#13 Iguais ou o Mesmo? — equals, hashCode e Identidade
#14 Dados sem Cerimônia — Records e Enums de Verdade
#15 Hierarquias Fechadas — Sealed Classes e Pattern Matching

Perguntas conceituais

Q20 Qual é a diferença entre uma **classe** e um **objeto**? Onde ficam, na memória da JVM, o objeto criado com `new` e a variável local que aponta para ele?

Q21 Quais são os quatro níveis de acesso do Java, do mais aberto ao mais fechado? Qual é o nível que se aplica quando nenhum modificador é escrito?

Q22 Por que **encapsular** um campo (deixá-lo `private` e expor métodos) é melhor do que deixá-lo `public`, se no fim o valor continua podendo ser lido? Use uma conta bancária como exemplo.

Q23 Em que ordem acontecem as etapas quando se executa `new ContaCorrente(...)`, sendo `ContaCorrente` subclasse de `ContaBancaria`? Por que a chamada `super(...)` precisa ser a primeira instrução do construtor (até o Java 24)?

Q24 Qual é o perigo de chamar um método que pode ser **sobrescrito** de dentro de um construtor? Dê um exemplo em que o resultado surpreende.

Q25 Quando escolher uma **classe abstrata** e quando escolher uma **interface**? Cite duas coisas que só a classe abstrata pode ter.

Q26 Uma classe implementa duas interfaces que têm o mesmo método `default void saudar()`. O que o compilador faz? Como resolver chamando a versão de uma delas?

Q27 O que é **polimorfismo**? Explique a diferença entre o tipo **declarado** de uma variável e o tipo **real** do objeto, e qual deles decide o método executado.

Q28 Quais são as regras do contrato entre `equals` e `hashCode`? O que acontece com um `HashSet` se uma classe sobrescreve `equals` e esquece o `hashCode`?

Q29 Por que é perigoso usar um objeto **mutável** como chave de um `HashMap` e depois alterar um campo que entra no `hashCode`?

Q30 O que o compilador gera automaticamente num `record`? E como validar os dados na criação, por exemplo impedindo um `Intervalo` com início maior que o fim?

Q31 Um `record` com um componente `List<String> itens` é realmente imutável? O que precisa ser feito para que seja?

Q32 Cite três coisas que um `enum` do Java pode ter e que as constantes inteiras (`static final int`) não podem. Para que serve o `EnumMap`?

Q33 O que uma interface `sealed` com `permits` garante? Por que ela combina tão bem com `switch` e *record patterns*?

Exercícios de código

Q34 Modele expressões aritméticas com uma interface selada `Expr` e os *records* `Num`, `Soma` e `Mult`. Escreva `avaliar(Expr)` com um `switch` sobre *record patterns*, e `mostrar(Expr)` que produza texto com parênteses. Teste com (2 + 3) × 4.

Q35 Use um `switch` com **guarda** (`when`) para classificar um valor `Object`: `Integer` negativo, `Integer` não negativo, `String` vazia, `String` com texto, e "outro". O que acontece se o `case Integer i` sem guarda vier **antes** do `case Integer i when i < 0`?

Q36 Escreva um `enum Operacao` com as constantes `SOMA`, `SUB`, `MULT` e `DIV`, cada uma com um símbolo (campo) e implementando o método `double aplicar(double a, double b)`. Imprima uma tabela aplicando todas a `8` e `2`.

Q37 Encontre e explique o erro de projeto: a classe abaixo quebra a **simetria** do `equals`. Mostre um caso em que `a.equals(b)` e `b.equals(a)` discordam.

```
class Ponto {
    final int x, y;
    Ponto(int x, int y) { this.x = x; this.y = y; }
    @Override public boolean equals(Object o) {
        return o instanceof Ponto p && p.x == x && p.y == y;
    }
    @Override public int hashCode() { return 31 * x + y; }
}
```

```
class PontoColorido extends Ponto {
    final String cor;
    PontoColorido(int x, int y, String cor) { super(x, y); this.cor = cor; }
    @Override public boolean equals(Object o) {
        return o instanceof PontoColorido p && super.equals(p) && p.cor.equals(cor);
    }
}
```

Q38 Transforme a classe mutável abaixo numa classe **imutável** correta e explique cada mudança.

```
public class Periodo {
    public java.util.Date inicio;
    public java.util.Date fim;
    public Periodo(java.util.Date inicio, java.util.Date fim) { this.inicio = inicio; this.fim =
fim; }
}
```

FASE 3

A biblioteca padrão

Artigos 16 a 22

#16 Escreva uma Vez para Todo Tipo — Genéricos
#17 O Preço da Compatibilidade — Wildcards e Type Erasure
#18 Listas, Conjuntos e Mapas — o Collections Framework
#19 Ordem e Critério — Comparable, Comparator e Coleções Ordenadas
#20 Quando o Contrato se Quebra — Exceções Verificadas e Não Verificadas
#21 Recursos que se Fecham Sozinhos — try-with-resources, Arquivos e NIO.2
#22 Tempo, Dinheiro e Padrões — java.time, BigDecimal e Expressões Regulares

Perguntas conceituais

Q39 Que problema os genéricos resolveram? Compare uma `List` "crua" (*raw type*) com uma `List<String>` quanto ao momento em que um erro de tipo é descoberto.

Q40 O que é **type erasure**? Cite duas coisas que você **não** consegue fazer com um parâmetro de tipo `T` por causa dele.

Q41 Explique a regra **PECS** (*Producer Extends, Consumer Super*) usando a assinatura de `Collections.copy(List<? super T> destino, List<? extends T> origem)`.

Q42 Por que `List<Integer>` **não** é subtipo de `List<Number>`, embora `Integer` seja subtipo de `Number`? Mostre o que daria errado se fosse.

Q43 Compare `ArrayList` e `LinkedList`: qual é o custo de `get(i)`, de inserir no fim e de inserir no meio? Qual das duas deve ser a escolha padrão, e por quê?

Q44 O que é uma `ConcurrentModificationException`? Por que ela aparece num programa **sem nenhuma thread**, e quais são as formas corretas de remover elementos enquanto se percorre uma lista?

Q45 Qual é a diferença entre `List.of(...)`, `Collections.unmodifiableList(lista)` e `List.copyOf(lista)`?

Q46 Para que servem as **Sequenced Collections** (Java 21)? Dê dois métodos novos e duas classes que passaram a implementá-las.

Q47 Qual é a diferença entre `Comparable` e `Comparator`? Por que escrever `return a.idade - b.idade;` num comparador é um erro, ainda que funcione quase sempre?

Q48 Explique a diferença entre exceções **verificadas** e **não verificadas**, e dê a regra de bolso de quando usar cada uma.

Q49 O que há de errado neste tratamento de exceção? Liste pelo menos três problemas.

```
try {
    processarPedido(pedido);
} catch (Exception e) {
    // TODO
}
```

Q50 O que são **exceções suprimidas** no `try` com recursos? Em que situação elas aparecem?

Q51 Qual é a diferença entre `Instant`, `LocalDateTime` e `ZonedDateTime`? Qual deles usar para gravar **quando** um evento aconteceu?

Q52 Por que `new BigDecimal(0.1)` e `BigDecimal.valueOf(0.1)` dão resultados diferentes? E por que `new BigDecimal("2.0").equals(new BigDecimal("2.00"))` é `false`?

Exercícios de código

Q53 Escreva um método genérico `static <T extends Comparable<? super T>> T maximo(List<? extends T> lista)` que devolva o maior elemento e lance `NoSuchElementException` para lista vazia. Explique cada parte da assinatura.

Q54 Conte a frequência das palavras de um texto com um `Map<String, Integer>` usando `merge`, e imprima as 3 mais frequentes, com empates desempatados por ordem alfabética.

Q55 Crie a exceção verificada `SaldoInsuficienteException`, que guarda o saldo e o valor pedido, e use-a num método `sacar`. Mostre o tratamento no chamador, imprimindo uma mensagem útil.

Q56 Usando NIO.2 (`Files` e `Path`), escreva um programa que crie a pasta `saida`, grave nela um arquivo `diario.txt` com três linhas, depois leia o arquivo com `Files.lines` dentro de um `try` com recursos, contando as linhas que contêm "Java".

Q57 Calcule as datas de vencimento de 3 parcelas mensais a partir de 31/01/2026 e diga quantos dias faltam da primeira até a última. Explique o que o `plusMonths` faz com o dia 31.

Q58 Escreva uma expressão regular com **grupos nomeados** que extraia dia, mês e ano de datas no formato `dd/mm/aaaa` dentro de um texto, e imprima cada uma no formato ISO (`aaaa-mm-dd`).

FASE 4

Java funcional

Artigos 23 a 27

#23 Comportamento como Valor — Lambdas e Interfaces Funcionais
#24 Pare de Escrever Laços — a API de Streams
#25 Agrupar, Achatar, Reduzir — Collectors e Gatherers
#26 Um Valor que Pode Não Existir — Optional e a Guerra contra o null
#27 O Programa que se Examina — Anotações e Reflexão

Perguntas conceituais

Q59 O que é uma **interface funcional**? Associe cada uma destas interfaces à forma da função que ela representa: `Supplier<T>`, `Consumer<T>`, `Function<T, R>`, `Predicate<T>`, `UnaryOperator<T>` e `BiFunction<T, U, R>`.

Q60 Por que uma *lambda* só pode usar variáveis locais que sejam **efetivamente finais**? Por que essa restrição não vale para campos?

Q61 Cite os quatro tipos de **referência de método** e dê um exemplo de cada.

Q62 Explique a **preguiça** (*laziness*) dos *streams*. Quantas vezes o `filter` é executado no código abaixo, e por quê?

```
var r = Stream.of("ana", "bia", "caio", "davi", "eva")
        .filter(n -> { IO.println("filtrando " + n); return n.length() == 4; })
        .findFirst();
```

Q63 Por que um *stream* não pode ser consumido duas vezes? E por que não se deve modificar uma variável externa (por exemplo, somar num campo) de dentro de um `forEach` ou `map`?

Q64 Qual é a diferença entre `map` e `flatMap`? Dê um exemplo em que só `flatMap` resolve.

Q65 O que são os **Gatherers** (Java 24, JEP 485)? Que tipo de operação eles permitem que antes era difícil com *streams*?

Q66 Liste três usos **errados** de `Optional` e diga o que fazer no lugar de cada um.

Q67 Qual é a diferença entre `orElse(criar())` e `orElseGet(() -> criar())`? Quando essa diferença importa?

Q68 O que é **retenção** de uma anotação (`SOURCE`, `CLASS`, `RUNTIME`)? Que retenção é obrigatória para que um *framework* leia a anotação por reflexão, e em que tipos de elemento ela pode ser posta (`@Target`)?

Q69 Cite duas desvantagens da **reflexão** que justificam usá-la com moderação no código de aplicação.

Exercícios de código

Q70 Dada uma lista de vendas (`record Venda(String vendedor, String regiao, double valor)`), use *streams* para obter: (a) o total por região, ordenado pelo nome da região; (b) o vendedor com o maior total; (c) as regiões separadas entre "acima" e "abaixo" de 1.000 em total.

Q71 Use `Gatherers.windowSliding` para calcular a **média móvel** de 3 dias de uma série de temperaturas, e `Gatherers.scan` para o total acumulado de uma lista de depósitos.

Q72 Reescreva o código abaixo usando `Optional` de forma idiomática, sem nenhum `if` e sem `get()`.

```
String cidadeDoCliente(long id) {
    Cliente c = repositorio.buscar(id);          // pode devolver null
    if (c != null) {
        Endereco e = c.getEndereco();            // pode ser null
        if (e != null && e.getCidade() != null) {
            return e.getCidade().toUpperCase();
        }
    }
    return "DESCONHECIDA";
}
```

Q73 Crie a anotação `@Positivo`, aplicável a campos, e um validador que use **reflexão** para conferir, em qualquer objeto, se os campos `int` anotados são maiores que zero, devolvendo a lista de violações.

FASE 5

Projetos e ferramentas

Artigos 28 a 32

#28 Do Arquivo Solto ao Projeto — Pacotes, Classpath e JARs
#29 Construção Automatizada — Maven e Gradle
#30 Fronteiras Explícitas — o Sistema de Módulos
#31 Provando que Funciona — Testes com JUnit e AssertJ
#32 Dublês e Disciplina — Mockito e Testes que Não Mentem

Perguntas conceituais

Q74 Qual é a relação entre o nome do **pacote** e a estrutura de pastas? O que é o **classpath**, e o que acontece quando duas bibliotecas no classpath têm uma classe com o mesmo nome completo?

Q75 O que o arquivo `META-INF/MANIFEST.MF` de um JAR executável precisa ter? Qual é a diferença entre um JAR comum e um "JAR gordo" (*fat jar*)?

Q76 Explique as **fases** do ciclo de vida padrão do Maven que você mais usa (`compile`, `test`, `package`, `verify`, `install`) e o que significa rodar `mvn package`.

Q77 Qual é a diferença entre os escopos de dependência `compile`, `test`, `provided` e `runtime` no Maven (ou `implementation`, `testImplementation`, `compileOnly` e `runtimeOnly` no Gradle)?

Q78 No `module-info.java`, o que fazem as diretivas `requires`, `requires transitive`, `exports` e `opens`? Por que *frameworks* como JPA e Jackson precisam de `opens`?

Q79 O que é o **encapsulamento forte** que o sistema de módulos trouxe? Por que `sun.misc.Unsafe` e as classes internas do JDK deixaram de ser acessíveis por padrão?

Q80 O que é a pirâmide de testes? Por que testes **unitários** devem ser a maioria, e o que um teste de **integração** verifica que o unitário não verifica?

Q81 O que torna um teste **"mentiroso"**? Cite três sinais de que um teste passa sem provar nada.

Q82 Qual é a diferença entre um *stub*, um *mock* e um *spy* no Mockito? Quando é preferível escrever um **fake** à mão?

Exercícios de código

Q83 Escreva uma classe `Calculadora` com o método `static int dividir(int a, int b)`, e uma classe de testes JUnit 5 que verifique uma divisão normal, a divisão por zero (esperando `ArithmeticException`) e vários casos com `@ParameterizedTest`.

Q84 O serviço abaixo envia um e-mail de boas-vindas ao cadastrar um cliente novo, mas não envia se o e-mail já existir. Escreva os testes com Mockito para os dois cenários.

```
interface Repositorio { boolean existe(String email); void salvar(String email); }
interface Correio { void enviar(String para, String assunto); }
```

```
class Cadastro {
    private final Repositorio repo;
    private final Correio correio;
    Cadastro(Repositorio repo, Correio correio) { this.repo = repo; this.correio = correio; }
```

```
    boolean cadastrar(String email) {
        if (repo.existe(email)) return false;
        repo.salvar(email);
        correio.enviar(email, "Bem-vindo!");
        return true;
    }
}
```

Q85 Monte um projeto modular com dois módulos: `br.com.exemplo.tarefas` (o modelo, com o pacote `br.com.exemplo.tarefas`) e `br.com.exemplo.tarefas.app` (a aplicação que usa o modelo). Escreva os dois `module-info.java` e os comandos `javac` e `java` para compilar e executar.

FASE 6

Por dentro da JVM

Artigos 33 a 36

#33 A Máquina por Trás da Máquina — Class Loading, Pilha, Heap e JIT
#34 Quem Recolhe o Lixo — os Coletores G1 e ZGC
#35 Medir Antes de Otimizar — JFR, Mission Control e JMH
#36 Ponte para o Nativo — a Foreign Function & Memory API

Perguntas conceituais

Q86 Descreva as três etapas do **carregamento de classes** (carregar, ligar e inicializar). Quando exatamente o bloco `static { ... }` de uma classe é executado?

Q87 O que fica na **pilha** de uma *thread* e o que fica no **heap**? Por que uma variável local nunca tem problema de concorrência, mas o objeto para o qual ela aponta pode ter?

Q88 O que é **análise de escape** (*escape analysis*)? Dê um exemplo de objeto que o JIT pode eliminar completamente.

Q89 Explique a **hipótese geracional** e por que ela torna a coleta de lixo barata para objetos de vida curta.

Q90 Compare o **G1** e o **ZGC**: como cada um trata as pausas? Em que tipo de aplicação você escolheria cada um?

Q91 Um programa Java pode ter **vazamento de memória**, mesmo com coletor de lixo? Dê dois exemplos clássicos.

Q92 Por que um *micro-benchmark* escrito com `System.nanoTime()` num laço costuma mentir? Cite três armadilhas que o **JMH** resolve.

Q93 O que o **JDK Flight Recorder** (JFR) registra, e por que ele pode ficar ligado em produção? Escreva o comando para gravar 60 segundos de um programa já em execução.

Q94 Na **FFM API** (Foreign Function & Memory, final no Java 22), para que servem o `Arena`, o `MemorySegment` e o `Linker`? Por que ela é mais segura que o JNI?

Exercícios de código

Q95 Use a FFM API para chamar a função `strlen` da biblioteca C padrão com a *string* "Olá, FFM!" e explique por que o resultado não é 9.

Q96 Escreva um programa que provoque um `OutOfMemoryError` controlado, guardando arrays de 1 MB numa lista estática, e execute-o com `-Xmx64m`. Depois explique como o `-XX:+HeapDumpOnOutOfMemoryError` ajudaria a encontrar a causa num sistema real.

FASE 7

Concorrência

Artigos 37 a 43

#37 Fazendo Duas Coisas ao Mesmo Tempo — Threads
#38 O Cadeado e a Memória — synchronized, volatile e o Java Memory Model
#39 Não Crie Threads, Entregue Tarefas — ExecutorService e Futures
#40 Ferramentas de Precisão — Locks, Atomics e Coleções Concorrentes
#41 Encadeando o que Ainda Não Aconteceu — CompletableFuture
#42 Um Milhão de Threads — Virtual Threads
#43 Tarefas em Família — Structured Concurrency e Scoped Values

Perguntas conceituais

Q97 Qual é a diferença entre chamar `thread.run()` e `thread.start()`? E o que `join()` faz?

Q98 O que é uma **condição de corrida**? Por que `contador++` não é uma operação atômica, mesmo sendo uma linha só?

Q99 Explique as duas garantias do `synchronized`: **exclusão mútua** e **visibilidade**. Por que só `volatile` não resolve um contador compartilhado?

Q100 Quais são as quatro condições necessárias para um **deadlock**? Qual delas é a mais prática de quebrar, e como?

Q101 Por que criar uma `new Thread()` para cada tarefa é má ideia com *threads* de plataforma? O que um `ExecutorService` resolve, e por que ele precisa ser **desligado**?

Q102 Um `ThreadPoolExecutor` foi configurado com 4 *threads* e uma fila **limitada** de 100 posições. O que acontece com a tarefa número 105, se todas estão ocupadas? Quais são as políticas de rejeição disponíveis?

Q103 Quando usar `AtomicLong` e quando usar `LongAdder`? E por que `ConcurrentHashMap.merge(chave, 1, Integer::sum)` é seguro, mas `mapa.put(chave, mapa.get(chave) + 1)` não é, mesmo num `ConcurrentHashMap`?

Q104 Qual é a diferença entre `thenApply` e `thenCompose` num `CompletableFuture`? E como tratar um erro que aconteceu em qualquer etapa da cadeia?

Q105 O que é uma **virtual thread**? Por que ela permite um milhão de conexões simultâneas? Em que tipo de carga ela **não** traz ganho?

Q106 O que é o *pinning* de uma *virtual thread*? O que mudou no Java 24 em relação ao `synchronized`?

Q107 Que problemas a **concorrência estruturada** (`StructuredTaskScope`) resolve em relação a submeter duas tarefas num `ExecutorService` e chamar `get()` nelas?

Q108 O que é um `ScopedValue` e por que ele é preferível a um `ThreadLocal` com *virtual threads*?

Exercícios de código

Q109 Demonstre a perda de incrementos: 4 *threads* somam 1 num `int` comum 250.000 vezes cada. Depois corrija com `AtomicInteger` e com `LongAdder`, e mostre os três totais.

Q110 Com um `ExecutorService` de *virtual threads*, dispare 10.000 tarefas que "esperam a rede" por 100 ms (`Thread.sleep`) e devolvem o próprio número. Some os resultados e meça o tempo total. Explique por que leva perto de 100 ms, e não 1.000 segundos.

Q111 Encadeie com `CompletableFuture`: buscar o preço de um produto (assíncrono), buscar a cotação do dólar (assíncrono, **em paralelo**), combinar os dois para obter o preço em dólares, e cair num valor padrão se qualquer busca falhar.

Q112 Escreva uma transferência entre contas que **não** entre em *deadlock* quando duas *threads* transferem em sentidos opostos ao mesmo tempo (A→B e B→A), usando ordem global de *locks*.

Q113 Com `StructuredTaskScope` (*preview* no Java 25), busque ao mesmo tempo os dados de um usuário e os pedidos dele. Se uma das buscas falhar, a outra deve ser cancelada. Mostre o código e o comando para executá-lo.

FASE 8

Qualidade e distribuição

Artigos 44 a 46

#44 Lições de Quem Veio Antes — Effective Java, Código Limpo e Padrões
#45 Contratos à Vista — Imutabilidade, Programação Defensiva e JSpecify
#46 Do JAR ao Executável — jlink, jpackage e GraalVM Native Image

Perguntas conceituais

Q114 Por que *Effective Java* recomenda "prefira composição a herança"? Dê um exemplo em que herdar de uma classe da biblioteca quebra de forma sutil.

Q115 Quando um **builder** é melhor que um construtor? E quando um *record* com um construtor comum já basta?

Q116 O que é **programação defensiva**? Explique as cópias defensivas "na entrada" e "na saída" e o princípio *fail-fast*.

Q117 O que as anotações do **JSpecify** (`@NullMarked`, `@Nullable`) acrescentam ao código? Elas mudam o comportamento em tempo de execução?

Q118 Qual é a diferença entre o resultado do `jlink`, do `jpackage` e do **GraalVM Native Image**? Quando cada um é a escolha certa?

Q119 O que a opção `--generate-cds-archive` do `jlink` faz, e que problema ela ataca? Como o Projeto Leyden continua essa ideia?

Exercícios de código

Q120 Implemente um builder para `record Pedido(String cliente, List<String> itens, boolean urgente, String observacao)`, com cliente obrigatório, pelo menos um item, `urgente` falso e `observacao` vazia por padrão, validando tudo no `build()`.

Q121 Escreva os comandos para: (1) descobrir os módulos de que o JAR `app.jar` precisa; (2) gerar com `jlink` um runtime mínimo, sem cabeçalhos, páginas de manual e informações de depuração; (3) criar com `jpackage` um instalador `.deb` usando esse runtime.

FASE 9

O projeto final: o servidor HTTP Farol

Artigos 47 a 52

#47 O Projeto Final Começa — Arquitetando um Servidor HTTP sobre Sockets
#48 A Primeira Pedra — ServerSocket e a Leitura do HTTP/1.1
#49 A Segunda Pedra — Requisições e Respostas como Tipos: Records, Sealed e Rotas
#50 A Terceira Pedra — Mil Conexões ao Mesmo Tempo: Virtual Threads e Keep-Alive
#51 A Pedra de Acabamento — Arquivos Estáticos, Testes e Empacotamento
#52 A Jornada Completa — Retrospectiva de Um Ano de Java

Perguntas conceituais

Q122 Descreva a estrutura de uma requisição HTTP/1.1 como ela chega pelo *socket*: quais são as partes, como elas se separam, e como o servidor sabe onde o corpo termina?

Q123 O que é *request smuggling*, e por que o Farol recusa com 400 uma requisição que traz **ao mesmo tempo** `Content-Length` e `Transfer-Encoding`?

Q124 O que é *response splitting*? Que validação o `EscritorDeResposta` precisa fazer nos valores de cabeçalho, e por quê?

Q125 Como o Farol se protege de clientes lentos ou maliciosos? Cite os limites do *record* `Limites` e o código de status de cada um.

Q126 Por que o Farol tem **dois** prazos diferentes, o prazo de leitura e o prazo ocioso? O que aconteceria se houvesse um só?

Q127 No roteador do Farol, as rotas `/tarefas/{id}` e `/tarefas/novas` podem casar com o mesmo caminho. Como o `Roteador` escolhe? E quando ele devolve **405** em vez de **404**?

Q128 Explique as **duas** defesas do Farol contra *path traversal* em `ArquivosEstaticos`. Por que só verificar se o caminho contém ".." não basta?

Q129 Como funcionam `ETag` e `If-None-Match`? O que o servidor ganha ao responder `304 Not Modified`?

Q130 Como o Farol faz o **desligamento gracioso**? Por que simplesmente encerrar o processo é ruim para os clientes?

Q131 Na retrospectiva, o Farol usou praticamente todas as fases do curso. Associe cada item a uma fase: `record`s e `sealed` para requisição e resposta; `Semaphore` e *virtual threads*; `toRealPath` e `Files`; testes com `HttpClient`; `jlink` com CDS.

Exercícios de código

Q132 Escreva um método que interprete a **linha de pedido** HTTP (`"GET /tarefas?pagina=2 HTTP/1.1"`) num *record* `LinhaDePedido(String metodo, String caminho, String consulta, String versao)`, recusando linhas malformadas com `IllegalArgumentException`.

Q133 Implemente a decodificação de um corpo `Transfer-Encoding: chunked` a partir de uma `String`, respeitando um tamanho máximo de corpo. Cuidado com tamanhos em hexadecimal como `ffffffff`.

Q134 Escreva a verificação de *path traversal* com `toRealPath`: um método `Optional<Path> resolver(Path raiz, String pedido)` que devolve o arquivo só se ele existir e estiver **dentro** da raiz. Teste com `index.html`, `../segredo.txt` e um *link* simbólico que aponta para fora.

TRILHA A

Web, API e bancos de dados: as Trilhas da Ilha

Artigos 53 a 66

#53 Extensão — Java no Servidor: Servlets, Jakarta EE, Spring e Quarkus
#54 Extensão — Dados que Sobrevivem: SQL Essencial e JDBC com SQLite
#55 Extensão — Spring Boot pela Primeira Vez: Injeção de Dependência
#56 Extensão — Objetos no Banco: JPA e Hibernate
#57 Extensão — O Banco que Evolui: Spring Data e Migrations com Flyway
#58 Extensão — PostgreSQL: o Banco Servidor, JSONB e Arrays
#59 Extensão — MySQL: Driver, Dialeto e as Diferenças que Importam
#60 Extensão — MongoDB: Documentos, Agregações e Quando Escolher Cada Modelo
#61 Extensão — A API das Trilhas: REST, Validação, Erros e OpenAPI
#62 Extensão — Quem É Você? Spring Security e JWT
#63 Extensão — Bancos de Verdade nos Testes: Testcontainers
#64 Extensão — Páginas Dinâmicas: Thymeleaf e Formulários
#65 Extensão — O Site Completo: htmx, Login, Área Administrativa e Upload
#66 Extensão — Do Laptop ao Servidor: Docker, nginx e Deploy

Perguntas conceituais

Q135 Qual é a relação entre a API de **Servlets**, o **Jakarta EE**, o **Spring** e o **Quarkus**? Por que o pacote mudou de `javax.*` para `jakarta.*`?

Q136 Explique a diferença entre `INNER JOIN` e `LEFT JOIN` com as tabelas `trilha` e `ponto_de_interesse`. Qual delas lista também as trilhas sem nenhum ponto?

Q137 Por que concatenar texto do usuário numa consulta SQL é perigoso? Mostre o ataque com a entrada `' OR '1'='1` e explique por que o `PreparedStatement` o neutraliza.

Q138 O que é uma **transação**? Explique o que significa cada letra de **ACID** com o exemplo de uma inscrição numa trilha que decrementa as vagas e grava a inscrição.

Q139 O que é **injeção de dependência**? Por que o Spring prefere a injeção pelo **construtor** à injeção em campos com `@Autowired`?

Q140 Por que um método anotado com `@Transactional` que é chamado **por outro método da mesma classe** não abre transação?

Q141 Quais são os estados de uma entidade JPA (**transient**, **managed**, **detached**, **removed**)? O que é o *dirty checking*?

Q142 O que é o **problema N+1**? Mostre como ele aparece ao listar trilhas com os seus pontos de interesse, e duas formas de resolver.

Q143 Por que **nunca** se deve editar um *script* de migração do Flyway que já foi aplicado? Como se faz, então, para mudar uma coluna?

Q144 Cite três recursos do **PostgreSQL** usados nas Trilhas da Ilha que o SQLite não tinha (ou tinha de forma limitada), e para que cada um serviu.

Q145 Cite três diferenças práticas entre **MySQL** e **PostgreSQL** que obrigaram a ajustar o esquema ou o código das Trilhas.

Q146 Por que os **relatos** dos caminhantes foram para o **MongoDB**, enquanto o cadastro das trilhas ficou no PostgreSQL? Quando o modelo de documentos é a escolha certa?

Q147 Quais códigos de status HTTP a API das Trilhas devolve para: criar uma trilha com sucesso; buscar uma trilha que não existe; enviar JSON com um campo obrigatório vazio; fazer login com a senha errada; um usuário comum tentar apagar uma trilha; tentar criar uma trilha com um nome que já existe?

Q148 Descreva as três partes de um **JWT**. O conteúdo de um JWT é secreto? O que impede um usuário de trocar `"papel": "USUARIO"` por `"ADMIN"` no próprio *token*?

Q149 Por que senhas são guardadas com um *hash* **lento** e com **sal** (BCrypt, PBKDF2, Argon2), e não com SHA-256?

Q150 O que o **Testcontainers** resolve que um banco em memória (H2) não resolve? Por que o `@ServiceConnection` simplifica os testes?

Q151 Por que o **Thymeleaf** escapa o HTML por padrão em `th:text`? Quando `th:utext` seria aceitável, e que cuidado exige?

Q152 O que é **CSRF**? Por que ele afeta o site com login por sessão (*cookie*) mas não a API com JWT no cabeçalho `Authorization`?

Q153 O que o **htmx** muda na forma de fazer páginas interativas? Como ele se encaixa com o Thymeleaf no *back-end*?

Q154 Na imagem Docker das Trilhas, por que o `ENTRYPOINT` chama o `java` diretamente (forma *exec*, `["java", "-jar", "app.jar"]`) em vez de passar por um *shell* (`sh -c "java -jar app.jar"`)?

Q155 O que o **nginx** faz na frente da aplicação Spring no servidor? Cite três responsabilidades.

Q156 Onde devem ficar os **segredos** (senha do banco, chave do JWT) de uma aplicação em produção? Cite duas práticas erradas e uma certa.

Exercícios de código

Q157 Com JDBC puro e SQLite, crie a tabela `trilha (id, nome, km)`, insira três trilhas com `PreparedStatement` e mostre que a entrada maliciosa `' OR '1'='1` **não** retorna nada quando usada como parâmetro de busca por nome.

Q158 Escreva a consulta SQL que devolve, para cada dificuldade, a quantidade de trilhas e a média de quilômetros, mostrando só as dificuldades com pelo menos 2 trilhas. Qual é a diferença entre `WHERE` e `HAVING` nessa consulta?

Q159 Escreva um controlador REST do Spring para `GET /api/trilhas/{id}` e `POST /api/trilhas`, com um *record* de entrada validado (`nome` obrigatório, `km` positivo), devolvendo 201 com `Location` na criação e 404 quando a trilha não existe.

Q160 Escreva a consulta PostgreSQL que encontra as trilhas com a etiqueta `cachoeira` **e** que aceitam animais (`detalhes->>'pet' = 'true'` no JSONB), ordenadas pela relevância de uma busca textual por "mata atlântica". Quais índices ajudam essa consulta?

Q161 Escreva uma agregação do MongoDB (no `mongosh`) que calcule, para cada trilha, a nota média e a quantidade de relatos, só com os relatos de 2026, ordenando pela média e mostrando as 5 melhores.

Q162 Escreva um `Dockerfile` em **dois estágios** para a aplicação Spring Boot das Trilhas: um estágio que compila com Maven e outro, mínimo, que só roda. O contêiner não deve rodar como *root*.

TRILHA B

Desktop e Android

Artigos 67 a 70

#67 Extensão — Janelas em Java: AWT, Swing e JavaFX
#68 Extensão — JavaFX: Cenas, Controles e FXML
#69 Extensão — Um App Desktop que Consome a API (MVVM)
#70 Extensão — Java no Android: o que Mudou com o Kotlin e um App que Consome a API

Perguntas conceituais

Q163 O que é a **thread de eventos** de uma interface gráfica (a EDT no Swing, a *JavaFX Application Thread* no JavaFX)? Quais são as duas regras de ouro para usá-la?

Q164 Por que o JavaFX deixou de vir no JDK a partir do Java 11, e como ele é usado hoje num projeto?

Q165 O que são **propriedades** e **bindings** no JavaFX? Dê um exemplo em que um *binding* substitui um ouvinte (*listener*) escrito à mão.

Q166 Explique o padrão **MVVM** usado no Painel das Trilhas: o que fica no *Model*, na *View* e no *ViewModel*? Por que o *ViewModel* pode ser testado sem abrir nenhuma janela?

Q167 Por que um app Android **não** pode chamar a API das Trilhas em `http://localhost:8080`, e qual endereço usar no emulador?

Q168 Por que, em 2026, a recomendação do Google para apps Android novos é Kotlin, e o que continua valendo do Java nesse ecossistema?

Exercícios de código

Q169 Escreva um **ViewModel** de busca de trilhas, com as propriedades `termo`, `carregando` e `mensagem`, e a lista `resultados`, que chama um cliente assíncrono e atualiza tudo na *thread* da interface. Mostre como seria o teste dele com um cliente *fake*.

Q170 Escreva o cliente HTTP que busca uma página de trilhas na API com o `java.net.http.HttpClient`, de forma assíncrona e com *timeout*, e converte o JSON em *records* com Jackson. O mesmo código funcionaria no Android?

TRILHA C

Jogos com libGDX

Artigos 71 a 77

#71 Um Novo Território — Java nos Jogos, o Game Loop e o Caso Minecraft
#72 O Primeiro Sprite — Janela, Desenho e Input com libGDX
#73 Quando as Coisas se Tocam — Colisão e o Primeiro Jogo: Asteroids
#74 Gravidade de Verdade — Física com Box2D
#75 Mundos em Blocos — Mapas de Tiles e um Jogo de Plataforma
#76 Sem Lixo no Loop — Pools, ECS e o Coletor nos Jogos
#77 O Capstone dos Jogos — o Plataforma Completo e Exportado

Perguntas conceituais

Q171 O que é o **game loop**? Por que um jogo não pode simplesmente esperar por eventos, como um servidor ou uma janela de formulário?

Q172 Por que a física do jogo deve avançar em **passo fixo** (1/60 s), com um acumulador, em vez de usar o tempo do quadro diretamente? Para que serve a **interpolação** na hora de desenhar?

Q173 Por que o libGDX exige que `Texture`, `SpriteBatch`, `BitmapFont` e o `World` do Box2D recebam `dispose()`, se o Java tem coletor de lixo?

Q174 Por que se trabalha em **metros** (unidades do mundo) e não em pixels, e o que o `FitViewport` faz com uma janela de proporção diferente?

Q175 Por que o teste de colisão de um tiro rápido com um asteroide pequeno usou **segmento contra círculo** (o caminho do passo inteiro), e não só o ponto final do tiro?

Q176 Qual é a diferença entre corpos **estáticos**, **cinemáticos** e **dinâmicos** no Box2D? Dê um exemplo de cada no jogo do cais.

Q177 Explique o **coyote time** e o **jump buffer** do jogo de plataforma. Por que eles são regras de *game design*, e não de física?

Q178 Por que a altura do pulo no jogo foi 2,3333 m e não os 2,45 m da fórmula $v_0^2/2g$? Que consequência isso tem para o *level design*?

Q179 Por que o carregador TMX do libGDX não encontrava objetos marcados no campo **"Class"** do Tiled 1.9? Como o jogo resolveu?

Q180 No teste com 30 mil partículas, a versão com objetos novos teve média de 0,37 ms por quadro, mas um pior quadro de quase 6 ms, caído numa pausa do coletor. Por que o **pior quadro** importa mais que a média num jogo, e o que as versões com `Pool` e com arrays mudaram?

Q181 Cite as três armadilhas de um `Pool` que o curso demonstrou e a regra única que evita todas.

Q182 No capstone, por que `setScreen` do `Game` **não** chama `dispose()` na tela antiga, e por que a troca de tela precisava ser a **última** linha do `render`?

Q183 Quais são os três degraus para distribuir o jogo, do `.jar` ao programa que roda sem Java instalado? O que cada um resolve?

Exercícios de código

Q184 Escreva um game loop de **passo fixo com acumulador** em Java puro (sem libGDX), que simule uma bola caindo sob gravidade de $-9{,}8\ m/s^2$ durante 1 segundo "real", recebido em quadros de tamanhos irregulares (0,016 s, 0,033 s, 0,005 s...). Mostre que o resultado depende só do número de passos, e não do tamanho dos quadros.

Q185 Implemente a resolução de colisão de uma caixa (AABB) contra uma **grade de tiles**, um eixo de cada vez: mova em X, corrija se entrou em bloco sólido, e depois faça o mesmo em Y, marcando `noChao` quando a correção for para cima.

Q186 Escreva um `Pool` genérico mínimo, com `obtain()`, `free()` e um contador de objetos criados. Use-o para simular 1.000 quadros em que 50 balas nascem e 50 morrem por quadro, e mostre quantos objetos foram criados de verdade.

ENCERRAMENTO

O destino e o novo começo

Artigo 78

#78 O Destino e o Novo Começo — Encerrando a Jornada Completa

Perguntas de revisão

Q187 A última aula listou seis princípios que atravessaram o curso inteiro. Escolha **três** e dê, para cada um, um exemplo concreto de **duas trilhas diferentes** em que ele apareceu.

Q188 Por que o Java 25 é a escolha certa para um projeto que vai para produção hoje, mesmo com o JDK 27 já lançado? Qual é o ritmo de lançamentos do Java?

Q189 Monte um plano de estudos pessoal para os próximos 90 dias com uma meta concreta por mês, escolhendo **um** destino (*back-end* e nuvem, JVM por dentro, Android, jogos ou código aberto). Para cada mês, diga como você vai saber que cumpriu a meta.

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam várias fases e trilhas. Não há uma única resposta certa: o gabarito mostra um caminho.

D1

O Farol com HTTPS e compressão

Estenda o servidor Farol (Fase 9) para: (a) aceitar conexões TLS com `SSLServerSocket` e um certificado autoassinado gerado com `keytool`; (b) comprimir com gzip as respostas de texto quando o cliente enviar `Accept-Encoding: gzip`, acrescentando `Content-Encoding` e `Vary`; (c) não comprimir respostas pequenas (menos de 1 KB) nem imagens; (d) manter os testes existentes passando e acrescentar testes com `HttpClient` para os dois recursos.

Sua solução (arquitetura, decisões e código principal):

D2

Inscrições concorrentes sem vender vaga a mais

Nas Trilhas da Ilha, uma trilha guiada tem 10 vagas. Escreva o serviço de inscrição (Spring + PostgreSQL) e um teste de integração com Testcontainers que dispare 50 inscrições **simultâneas** (com *virtual threads*) e prove que exatamente 10 são aceitas e 40 recebem **409 Conflict**. Compare três estratégias: `SELECT ... FOR UPDATE`, um `UPDATE` atômico condicional e o bloqueio otimista com `@Version`.

Sua solução (arquitetura, decisões e código principal):

D3

Painel desktop com atualização ao vivo

Estenda o Painel JavaFX (Trilha B) para mostrar as inscrições de uma trilha **em tempo real**: a API passa a expor um *endpoint* de *Server-Sent Events* (`text/event-stream`) que emite um evento a cada inscrição, e o Painel consome o fluxo com `HttpClient` (`BodyHandlers.ofLines()`), atualizando uma lista e um contador de vagas sem travar a interface. Trate a queda de conexão com reconexão e espera exponencial.

Sua solução (arquitetura, decisões e código principal):

D4

Uma terceira fase com inimigos

Acrescente ao jogo *A Ilha dos Blocos* (Trilha C) uma terceira fase com **inimigos** que andam de um lado para o outro nas plataformas e viram na borda ou numa parede. Encostar num inimigo pela lateral faz o herói voltar ao início, e pular em cima dele o elimina e dá um pequeno impulso para cima. A regra precisa estar na `Fase`, testada sem GPU, e o piloto automático precisa continuar vencendo **todas** as fases.

Sua solução (arquitetura, decisões e código principal):

D5

Métricas do Farol com JFR e um painel

Instrumente o Farol com **eventos JFR personalizados** (`jdk.jfr.Event`): um evento por requisição com método, rota, status e duração. Grave uma sessão de carga (gerada com um programa de *virtual threads* fazendo 100 mil requisições) e escreva um analisador, com a API `jdk.jfr.consumer.RecordingFile`, que produza a tabela de latência p50/p95/p99 por rota e a taxa de erros. Compare os números com o servidor usando o G1 e o ZGC.

Sua solução (arquitetura, decisões e código principal):

D6

A jornada completa em um só sistema

Integre as três trilhas: o jogo envia a pontuação de cada fase vencida para a API das Trilhas (autenticado com JWT); a API valida a pontuação contra trapaças (tempo mínimo possível para a fase, moedas no máximo o total da fase, assinatura do jogo), grava no PostgreSQL e mantém um *ranking*; o site mostra o *ranking* ao vivo com htmx; e o Painel desktop permite a um administrador anular pontuações suspeitas. Escreva um documento de arquitetura de uma página e implemente pelo menos o caminho jogo → API → site.

Sua solução (arquitetura, decisões e código principal):

D7

Uma calculadora, duas interfaces

Escreva uma calculadora com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada. Ela deve: (a) fazer as contas com `BigDecimal`, para que `0,1 + 0,2` dê exatamente `0,3`, aceitando vírgula ou ponto e limitando o resultado a 16 dígitos significativos; (b) guardar o último resultado em `ans`, que pode ser usado na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário e texto que não é número com uma mensagem clara, **sem encerrar o programa**. Escreva o núcleo numa classe que não sabe nada de entrada e saída, e reaproveite-o em duas interfaces: uma de **terminal** (com os comandos `hist` e `sair`) e uma **gráfica** em JavaFX, com teclado na tela, visor e histórico.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!