

ESPECIALISTA
JAVA

De Programador a Especialista em Java e Padrões

Supere o medo de trabalhar com Java e mergulhe fundo na linguagem, principais APIs do JDK e padrões de mercado com a escola referência #1 no Brasil

algaworks

O roadmap perfeito que vai te capacitar para trabalhar com Java nas maiores empresas



Fundamentos sólidos do Java

Programar em Java sem conhecer os fundamentos da linguagem é como pilotar um avião sem conhecer os fundamentos de aerodinâmica e navegação aérea. Talvez você até consiga decolar uma aeronave sem isso, mas dificilmente conseguirá pousar em segurança.

No EJ (Especialista Java) você vai aprender tópicos como declaração e inicialização de variáveis, tipos primitivos, conversão de tipos primitivos e promoção aritmética.

E vai aprender diversos tipos de operadores, como operadores aritméticos, abreviação de operadores, operadores de incremento e decremento, operadores de igualdade e de negação (unário), operadores de comparação, operadores lógicos, curto-circuito de operadores lógicos, precedência de operadores lógicos, operador ternário, etc.

Além disso, vai aprender a receber entrada de dados do usuário com Scanner, formatação da saída com printf e como usar JShell (REPL do Java) para rodar códigos Java.

E claro, você vai aprender as estruturas de controle também, como a estrutura condicional if, else e else if, switch, cláusula break, switch expressions, estrutura de repetição for, while e do/while, etc.

Orientação a objetos ensinada como mágica

O paradigma de programação da orientação a objetos é lindo de ver funcionando, mas até a ficha cair, muitas pessoas desistem de aprender.

No EJ, as aulas de orientação a objetos (OO) são densas, porém tudo muito mastigado, passo a passo e com vários exemplos.

Depois que você aprender OO, vai achar que foi mágica, mas a realidade é que é apenas o nosso método de ensino, que simplifica assuntos complexos de forma que fique suave e divertido aprender.

Você vai definitivamente dominar assuntos como classes, objetos, membros de instância e membros de classe, sobrecarga e sobrescrita de métodos, construtores, pacotes, visibilidade, encapsulamento, JavaBeans, herança, polimorfismo, classes abstratas, interfaces e muito mais.

Vai aprender também alguns recursos mais novos da linguagem, como Records e classes seladas.

Diagrama de classes da UML

Para discutir ideias e modelos de domínio ou arquitetura, o diagrama de classes da UML é muito útil até os dias de hoje.

Você vai aprender como interpretar e desenhar diagramas de classes enquanto aprende orientação a objetos, usando a ferramenta StarUML.

Produtividade com IntelliJ IDEA

No início, começamos programando com um editor de textos simples, para você se habituar com a sintaxe da linguagem, mas logo nós começamos a usar a IDE mais completa e mais usada no mercado: IntelliJ IDEA.

Você vai conhecer os principais recursos e atalhos da IDE e com certeza será muito mais produtivo depois de praticar as aulas sobre esse assunto.

Inclusive, a ferramenta de debug e o plugin Java Visualizer do IntelliJ IDEA será muito útil para você aprender orientação a objetos de um jeito mais visual, enxergando os objetos instanciados e as instruções em tempo de execução.

Além disso, você também vai aprender a usar Scratch Files para rascunhar códigos, JShell Console e EditorConfig, para garantir a consistência no estilo de codificação.

Operações com números e datas

Você vai aprender a trabalhar com números de forma correta, fazer comparações sem cair em armadilhas, trabalhar com operações matemáticas com `java.lang.Math`, usar `BigDecimal` para aplicações que exigem boa precisão, formatar com `DecimalFormat`, etc.

E também, vai aprender a trabalhar com data e hora, usando o tipo `Date`, `Calendar` e Date and Time API (pacote `java.time`). Vai aprender a usar os tipos `Instant`, `LocalDate`, `LocalTime`, `LocalDateTime`, `Period`, `Duration` e muito mais.

E claro, você vai aprender como fazer operações e comparações com data/hora também.

Expressões Lambda e Streams API

Você vai aprender a trabalhar com o paradigma da programação funcional em Java usando as Expressões Lambda, Method Reference e vai também implementar as suas próprias Interfaces Funcionais.

Com esses poderosos recursos, você escreve menos linhas de código, que ficam mais simples, mais expressivos e ainda evita erros bobos e fica muito mais produtivo!

E aproveitando os benefícios das Expressões Lambda, você também vai aprender como usar a Streams API para manipular coleções de maneira mais simples e eficiente.

Java Code Conventions

A linguagem Java possui convenções (padrões) de código bem definidos, usados por desenvolvedores no mundo inteiro.

Seguir e usar essas convenções não é só importante para manter o código elegante e legível, mas também é importante para que você seja reconhecido como um programador Java profissional (ou seja, se você não usar as convenções, provavelmente será visto como um amador).

Inclusive, aderir às convenções da linguagem é uma boa prática recomendada por autores importantes, como Joshua Bloch, em seu livro *Effective Java*.

E por ser algo tão relevante, os códigos dos exemplos das aulas serão escritos usando convenções de código, sempre sendo destacado a importância e o que é considerado certo e errado, para que você já aprenda Java escrevendo código compatível com o que o mercado espera de você.

Regado a princípios de Clean Code

Clean Code, ou Código Limpo, é uma forma de programar usando conceitos e técnicas simples para escrever códigos limpos e legíveis, ou seja, códigos facilmente compreendidos por qualquer desenvolvedor. Este método foi documentado por Robert Martin no livro Clean Code.

Neste curso você aprenderá a escrever código Java usando Clean Code desde o início, com aulas práticas para refatoração de código (para torná-lo "limpo"), além de aulas baseadas nos principais princípios recomendados por Robert Martin.

Você sairá na frente da maioria das pessoas, que só aprendem sobre Clean Code depois de alguns anos que já estão trabalhando com Java.

Técnicas e boas práticas do Effective Java

Java Efetivo, ou Effective Java, é o nome de várias regras (boas práticas) recomendadas por Joshua Bloch para escrever código Java de forma mais elegante e eficiente.

Você vai aprender diversas dessas boas práticas e com certeza seu código ficará em um nível de qualidade muito similar aos melhores programadores do mercado, que já possuem muitos anos de experiência.

As pessoas não vão acreditar que você é um iniciante em Java, quando verem o seu código! 😂

Exceptions do jeito certo

Quando algo inesperado acontece, uma exceção deve ser lançada.

Você vai conhecer a hierarquia das exceções, a diferença entre checked exceptions e unchecked exceptions e ainda entender a pilha de execução do Java.

Além disso, vai também aprender a capturar exceções com try/catch (incluindo multi-catch), lançar e propagar exceções e criar exceções customizadas.

E claro, tudo isso usando as boas práticas e com alerta das más práticas, para evitar cair nas armadilhas comuns que o mau uso das exceções pode gerar.

Arquivos JAR e Apache Maven

Você vai aprender como gerar e usar arquivos JAR como bibliotecas e JARs executáveis, para distribuir seus programas Java.

Além disso, também vai aprender a criar projetos com Apache Maven, uma ferramenta de gerenciamento de dependências e construção de projetos Java.

Logging com Logback e SLF4J

Logback é uma biblioteca muito usada para logging de aplicações Java, ou seja, para registrar eventos importantes que acontecem durante a execução de programas em algum local, como um arquivo em disco ou na nuvem. Essa biblioteca é considerada, inclusive, a sucessora do Log4j.

SLF4J é uma biblioteca que serve como abstração das diversas bibliotecas de logging, ou seja, é uma camada que "protege" o seu código, para evitar que ele tenha contato diretamente com bibliotecas específicas de logging.

Dessa forma, se um dia você precisar trocar a biblioteca de logging, fica super simples, já que o seu código só terá contato com a biblioteca SLF4J.

Você vai aprender a configurar um projeto Java com essas bibliotecas e configurar um File Appender (algo que gera logs em arquivos no disco).

Manipulação de arquivos e I/O

Você vai conhecer as APIs de I/O do Java e vai aprender, na prática, como manipular arquivos da maneira legada e também com NIO2.

Além disso, vai aprender sobre os tipos `BufferedReader`, `InputStream`, `PrintWriter`, `BufferedWriter`, `OutputStream` e `PrintStream`.

E também vai conhecer e usar o `try-with-resources` e implementar serialização e desserialização de objetos, além de entender o `serialVersionUID`.

Banco de dados e JDBC

No mundo corporativo, é muito comum precisar armazenar os dados em um banco de dados relacional.

Em Java, a API padrão para estabelecer uma conexão e manipular dados de bancos de dados relacionais é a JDBC.

Nós vamos usar o MySQL Server como nosso banco de dados, criar as tabelas, configurar um projeto Java com o driver JDBC do MySQL e executar consultas SQL e instruções DML.

Você vai aprender sobre os tipos Statement e PreparedStatement, além de aprender a trabalhar com transações do banco de dados.

E também, vai aprender sobre o padrão de projeto Repository, muito usado para isolar a camada de acesso ao banco de dados da camada de negócio da aplicação.

Ementa detalhada

1. Plataforma Java e ambiente de desenvolvimento

- 1.1. Introdução ao curso
- 1.2. Como aprender Java e pedir ajuda
- 1.3. Por que aprender Java?
- 1.4. Um pouco sobre a história do Java
- 1.5. Conhecendo as plataformas Java
- 1.6. Conhecendo a Máquina Virtual Java (JVM)
- 1.7. JRE e JDK: qual é a diferença?
- 1.8. Conhecendo as versões do Java
- 1.9. Conhecendo as distribuições de JDKs e licenças de uso
- 1.10. Instalando o JDK no Ubuntu e macOS com SDKMan!
- 1.11. Instalando o JDK no Windows
- 1.12. Escolhendo um editor de código simples

2. Fundamentos da linguagem Java

- 2.1. Criando o primeiro programa Java
- 2.2. Compilando e executando um programa Java
- 2.3. Desafio: correção de erros
- 2.4. Escrevendo comentários no código
- 2.5. Conhecendo e usando convenções de código
- 2.6. Palavras reservadas

- 2.7. Trabalhando com variáveis
- 2.8. Operadores aritméticos
- 2.9. Desafio: variáveis e operadores aritméticos
- 2.10. Abreviando operadores aritméticos
- 2.11. Operadores de incremento e decremento
- 2.12. Tipos primitivos: boolean, char, byte e short
- 2.13. Tipos primitivos: int e long
- 2.14. Tipos primitivos: float e double
- 2.15. Conversão de tipos primitivos
- 2.16. Desafio: tipos primitivos e conversão
- 2.17. Promoção aritmética
- 2.18. Desafio: promoção aritmética
- 2.19. Trabalhando com String
- 2.20. Usando sequências de escape
- 2.21. Formatando a saída com printf
- 2.22. Recebendo entrada de dados
- 2.23. Desafio: String, entrada de dados, printf, etc
- 2.24. Usando JShell: o REPL do Java

3. Estruturas de controle e operadores

- 3.1. Operadores de igualdade e de negação (unário)

- 3.2. Operadores de comparação
- 3.3. Operadores lógicos
- 3.4. Desafio: operadores de igualdade e lógicos
- 3.5. Curto-circuito de operadores lógicos
- 3.6. Precedência de operadores lógicos
- 3.7. Estrutura condicional if
- 3.8. Estruturas condicionais else e else if
- 3.9. Desafio: calculadora complexa de IMC
- 3.10. Escopos e inicialização de variáveis
- 3.11. Estrutura condicional switch
- 3.12. Cláusula break
- 3.13. Switch Expressions
- 3.14. Operador ternário
- 3.15. Desafio: estrutura switch e operador ternário
- 3.16. Estrutura de repetição for
- 3.17. Estrutura de repetição while
- 3.18. Estrutura de repetição do/while
- 3.19. Cláusulas break e continue
- 3.20. Desafio: estruturas de repetição

4. Produtividade com a IDE IntelliJ IDEA

- 4.1. Conhecendo as IDEs mais populares
- 4.2. Instalando e conhecendo a IntelliJ IDEA
- 4.3. Mais da IntelliJ IDEA: build, run, plugins, terminal e shared index
- 4.4. Usando Code Completion, Live Templates e Postfix Completion

- 4.5. Conhecendo os principais atalhos
- 4.6. Mais atalhos do IntelliJ IDEA
- 4.7. Usando o Debugger para depurar o seu código
- 4.8. Debugger: silenciamento, condição e desativação de breakpoints
- 4.9. Debugger: gerenciando variáveis e avaliando expressões
- 4.10. Debugger: watches e logging
- 4.11. Rascunhando e testando código com Scratch Files
- 4.12. Testando código com JShell Console da IDE
- 4.13. Consistência no estilo de codificação com EditorConfig
- 4.14. Importando um projeto existente na IDE

5. Mergulhando em orientação a objetos

- 5.1. O paradigma da Programação Orientada a Objetos (POO)
- 5.2. Entendendo o conceito de classes e objetos
- 5.3. Criando uma classe com atributos
- 5.4. Instanciando objetos
- 5.5. Acessando atributos de objetos
- 5.6. Conhecendo o diagrama de classes da UML
- 5.7. Desafio: instanciando objetos e acessando os atributos
- 5.8. Composição de objetos
- 5.9. Atribuindo o objeto na composição
- 5.10. Diagrama de classes: associação, agregação e composição
- 5.11. Valores padrão e inicialização de variáveis de instância
- 5.12. Inicialização de objetos em variáveis de instância
- 5.13. Caindo a ficha: variáveis referenciam objetos

- 5.14. Criando e invocando um método
- 5.15. Implementando a lógica do método
- 5.16. IntelliJ IDEA: debug de chamadas de métodos
- 5.17. Métodos com retorno
- 5.18. Implementando métodos menores e evitando duplicação de código
- 5.19. Saindo do método com a cláusula return
- 5.20. Métodos que retornam objetos
- 5.21. Refatorando para tornar uma classe mais rica
- 5.22. Discutindo nome e responsabilidade da classe
- 5.23. Métodos com argumentos
- 5.24. Passando objetos como argumentos de métodos
- 5.25. Desafio: composição de objetos e métodos
- 5.26. Diagrama de classes: métodos e dependências
- 5.27. Métodos que alteram variável de instância
- 5.28. Métodos que alteram o valor de parâmetro do tipo primitivo
- 5.29. Métodos que alteram o estado de objeto recebido como parâmetro
- 5.30. Usando a palavra-chave this
- 5.31. Atributos de classe com o modificador static
- 5.32. Método de instância alterando variável estática
- 5.33. Métodos de classe (estáticos)
- 5.34. Método estático acessando membro de instância
- 5.35. Desafio: membros estáticos
- 5.36. Declarando constantes com static e final

- 5.37. Modificador final em variáveis locais
- 5.38. Sobrecarga de métodos
- 5.39. Inferência de tipo de variável local
- 5.40. Desafio: modificador final em variáveis locais
- 5.41. Desafio: sobrecarga de métodos
- 5.42. Desafio: inferência de tipo de variável local

6. Começando com boas práticas e código limpo

- 6.1. Boas práticas com Effective Java e Clean Code
- 6.2. Código Limpo: escolha bons nomes
- 6.3. Código Limpo: tamanho e organização de classes
- 6.4. Código Limpo: comentários no código
- 6.5. Código Limpo: métodos pequenos e que fazem só uma coisa
- 6.6. Código Limpo: pensando melhor nos argumentos de métodos
- 6.7. Boas práticas: valide os argumentos

7. Wrappers e boxing

- 7.1. Usando classes wrapper
- 7.2. Métodos de conversão
- 7.3. Autoboxing e unboxing
- 7.4. Comparando wrappers
- 7.5. Desafio: wrappers e boxing
- 7.6. Boas práticas: prefira tipos primitivos a wrappers

8. Trabalhando com arrays

- 8.1. Declarando e inicializando arrays
- 8.2. Acessando e atribuindo elementos de arrays
- 8.3. Iterando em arrays
- 8.4. Transformando arrays em representações em string
- 8.5. Ordenando arrays em ordem natural e reversa
- 8.6. Criando arrays de objetos
- 8.7. Iterando em arrays de objetos
- 8.8. Copiando e expandindo arrays
- 8.9. Removendo elementos de arrays
- 8.10. Desafio: arrays
- 8.11. Um pouco da ArrayList da Collections API
- 8.12. Desafio: ArrayList
- 8.13. Diagrama de classes: multiplicidade para arrays
- 8.14. Boas práticas: retorne arrays ou coleções vazias no lugar de null
- 8.15. Criando arrays multidimensionais
- 8.16. Iterando em arrays multidimensionais
- 8.17. Lendo os parâmetros da linha de comando
- 8.18. Criando métodos com argumentos variáveis com Varargs
- 8.19. Boas práticas: use varargs com cuidado
- 8.20. Desafio: varargs

9. Gerenciamento de memória do Java

- 9.1. Estrutura da memória da JVM
- 9.2. Call Stack, Stack Memory e Heap Memory
- 9.3. Informações da Memória Heap com a Runtime API

- 9.4. Configurando a Memória Heap da JVM
- 9.5. Garbage Collector
- 9.6. Inalcançabilidade de objetos
- 9.7. Quando esgota a Memória Heap: OutOfMemoryError
- 9.8. Boas práticas: remova referências de objetos não usados

10. Construtores, pacotes e visibilidade

- 10.1. Criando e chamando construtores
- 10.2. Construtores com parâmetros
- 10.3. Sobrecarga de construtores
- 10.4. Boas práticas: valide os argumentos de construtores também
- 10.5. Encadeamento de chamadas de construtores
- 10.6. Diagrama de classes: construtores
- 10.7. Desafio: construtores
- 10.8. Modificador final em variáveis de instância
- 10.9. Organizando as classes em pacotes
- 10.10. Importando classes de pacotes
- 10.11. Modificador de acesso public e default
- 10.12. Modificador de acesso private
- 10.13. Diagrama de classes: visibilidade public, package e private
- 10.14. Desafio: pacotes e modificadores de acesso
- 10.15. Importando membros estáticos (static import)
- 10.16. Múltiplas classes não-públicas em um único arquivo
- 10.17. Conhecendo a documentação JavaDoc do Java SE

11. Encapsulamento, JavaBeans e Records

- 11.1. O problema da falta de encapsulamento
- 11.2. Implementando encapsulamento
- 11.3. JavaBeans e métodos getters e setters
- 11.4. IntelliJ IDEA: gerando getters e setters
- 11.5. Desafio: encapsulamento e JavaBeans
- 11.6. Boas práticas: use métodos de acesso em classes públicas (incluindo Tell Don't Ask)
- 11.7. Código limpo: Lei de Demeter (incluindo Train Wreck)
- 11.8. Boas práticas: não permita instanciação com construtor privado
- 11.9. Boas práticas: crie cópias defensivas
- 11.10. Boas práticas: minimize a mutabilidade (incluindo Value Object)
- 11.11. Records
- 11.12. Diagrama de classes: Records

12. Herança

- 12.1. Conhecendo o projeto deste módulo
- 12.2. Criando classes etiquetadas (tagged classes)
- 12.3. Duplicando classes e isolando os comportamentos
- 12.4. Conhecendo herança e o relacionamento no diagrama de classes
- 12.5. Implementando herança
- 12.6. Sobrescrita de métodos
- 12.7. Modificador de acesso protected
- 12.8. Anotação @Override
- 12.9. Chamando método da superclasse com super

- 12.10. A classe Object
- 12.11. Invocando construtores da superclasse
- 12.12. Criando construtores com parâmetros na superclasse e subclasses
- 12.13. Boas práticas: sempre sobrescreva o método Object.toString
- 12.14. Modificador final em classes e métodos
- 12.15. Desafio: implementando herança
- 12.16. Sobrescrevendo o método Object.equals

13. Polimorfismo e classes abstratas

- 13.1. Upcasting de referências
- 13.2. O problema que polimorfismo resolve
- 13.3. Entendendo o polimorfismo
- 13.4. Downcasting de referências
- 13.5. Operador instanceof
- 13.6. Pattern Matching para o operador instanceof
- 13.7. Evitando o uso de instanceof
- 13.8. Criando um projeto de faturamento
- 13.9. Classes abstratas
- 13.10. Métodos abstratos
- 13.11. Desafio: polimorfismo e classes abstratas

14. Interfaces

- 14.1. Entendendo as interfaces
- 14.2. Criando a primeira interface
- 14.3. Implementando a primeira interface

- 14.4. Nova interface e injeção de dependências
- 14.5. Conhecendo o projeto da financeira
- 14.6. Quando herança de classes se torna um problema
- 14.7. Código mais flexível: refatorando para usar interfaces
- 14.8. Métodos default em interfaces
- 14.9. Classes abstratas com interfaces
- 14.10. Métodos privados em interfaces
- 14.11. Métodos estáticos em interfaces
- 14.12. Variáveis são estáticas e finais em interfaces
- 14.13. Implementando múltiplas interfaces
- 14.14. Herança de interfaces
- 14.15. Desafio: interfaces

15. Boas práticas de herança e interfaces

- 15.1. Rigidez do código com herança
- 15.2. Boas práticas: prefira composição em vez de herança de classes
- 15.3. Código frágil: alto acoplamento com herança
- 15.4. Boas práticas: reduzindo acoplamento com composição
- 15.5. Decorator Pattern: consolidando os conhecimentos
- 15.6. Boas práticas: projete interfaces com cuidado
- 15.7. Boas práticas: use interfaces apenas para definir tipos
- 15.8. Boas práticas: referencie objetos por suas interfaces

16. Exceções

- 16.1. Introdução às exceções

- 16.2. Lançando exceções
- 16.3. Stack Trace: interpretando e analisando exceções
- 16.4. Capturando exceções com try/catch
- 16.5. Relançando exceções e printStackTrace
- 16.6. Capturando exceções com múltiplos blocos catch
- 16.7. Hierarquia das exceções, checked e unchecked exceptions
- 16.8. Capturando checked exceptions
- 16.9. Criando exceções customizadas
- 16.10. Variáveis de instância em exceções customizadas
- 16.11. Lançando e propagando checked exceptions
- 16.12. Capturando exceções menos específicas (upcasting)
- 16.13. Capturando e lançando nova exceção
- 16.14. Boa prática: embrulhe a causa raiz
- 16.15. Capturando exceções com multi-catch
- 16.16. Usando a cláusula finally
- 16.17. IntelliJ IDEA: lançando exceções na ferramenta de debug
- 16.18. IntelliJ IDEA: adicionando Java Exception Breakpoints
- 16.19. Boas práticas: lance exceções ao invés de retornar null
- 16.20. Boas práticas: não engula exceções
- 16.21. Desafio: exceções

17. Generics

- 17.1. Introdução aos Generics
- 17.2. Implementando métodos genéricos
- 17.3. Delimitando tipos genéricos

- 17.4. Criando classes genéricas
- 17.5. Criando interfaces genéricas
- 17.6. Usando curingas para tipos desconhecidos
- 17.7. Desafio: Generics

18. Collections Framework

- 18.1. Por que mais uma API?
- 18.2. Introdução às listas e ao tipo List
- 18.3. Como funciona a ArrayList
- 18.4. Usando listas do tipo ArrayList
- 18.5. Iterando em lista com for tradicional
- 18.6. Usando listas com Generics
- 18.7. Localizando objetos em listas
- 18.8. Manipulando objetos da lista
- 18.9. Percorrendo a lista com Iterator
- 18.10. Percorrendo a lista com ListIterator
- 18.11. Percorrendo Iterables com enhanced for
- 18.12. LinkedList: mais performance na adição e remoção
- 18.13. Usando listas do tipo LinkedList
- 18.14. Vector: a lista thread-safe
- 18.15. Boas práticas: reduza o acoplamento usando o tipo da interface
- 18.16. Convertendo de lista para array
- 18.17. Ordenando listas pela ordem natural
- 18.18. Boas práticas: considere implementar a interface Comparable
- 18.19. Comparators: ordenando listas com outros critérios

- 18.20. Desafio: listas
- 18.21. Introdução aos conjuntos e ao tipo Set
- 18.22. Usando conjuntos do tipo HashSet
- 18.23. Tabelas de espalhamento (hash tables)
- 18.24. Implementando o método hashCode
- 18.25. Testando a implementação de hashCode com HashSet
- 18.26. Usando conjuntos do tipo TreeSet
- 18.27. Usando conjuntos do tipo LinkedHashSet
- 18.28. Desafio: conjuntos
- 18.29. Introdução aos mapas e ao tipo Map
- 18.30. Usando mapas dos tipos HashMap e Hashtable
- 18.31. Linked HashTable e TreeMap: outras implementações de mapas
- 18.32. Desafio: mapas
- 18.33. Boas práticas: encapsulamento com coleções não-modificáveis
- 18.34. Coleções imutáveis
- 18.35. Usando a API de List para manipular arrays

19. Enumerações

- 19.1. Usando enumerações à moda antiga
- 19.2. Criando tipos Enum
- 19.3. Diagrama de classes: enumeração
- 19.4. Usando os métodos do tipo Enum
- 19.5. Declarando e inicializando propriedades e construtores
- 19.6. Implementando métodos
- 19.7. Implementando métodos abstratos

19.8. Boas práticas: substitua parâmetros booleanos por enums

19.9. Desafio: enumerações

20. Trabalhando com strings

20.1. Comparação de strings

20.2. Pool de strings

20.3. Validando o conteúdo de strings

20.4. Extraindo trechos da String com indexOf e substring

20.5. Extraindo trechos da String com lastIndexOf e substring

20.6. Transformando strings

20.7. Implementando algoritmos usando os métodos da classe String

20.8. Desafio: validação de e-mail

20.9. Testando correspondências de strings com expressões regulares

20.10. Web Scraping: buscando correspondências com Pattern e Matcher

20.11. Manipulando strings com expressões regulares

20.12. Boas práticas: use StringBuilder para mais performance

20.13. Código mais limpo com Text blocks

20.14. Desafio: Text blocks e expressões regulares

21. Trabalhando com números

21.1. Comparando números da forma correta

21.2. Caching de classes wrapper

21.3. Operações matemáticas com java.lang.Math

21.4. Boas práticas: evite float e double se precisão é importante

21.5. Precisão nos cálculos com BigDecimal

21.6. Divisão de BigDecimal e formas de arredondamento

21.7. Formatação decimal com DecimalFormat

21.8. Localizando a formatação de números com Locale

21.9. Formatação numérica compacta

21.10. Transformando String em números com DecimalFormat

21.11. Desafio: formatação numérica

22. Date e Calendar: as APIs legadas

22.1. Entendendo os fusos horários

22.2. Instanciando datas com o tipo Date

22.3. Calculando e comparando datas com Date

22.4. Formatando Date para String

22.5. Convertendo de String para Date

22.6. Conhecendo o tipo Calendar

22.7. Obtendo unidades de tempo e atribuindo uma Date em Calendar

22.8. Operações de datas com o tipo Calendar

22.9. Comparando datas com o tipo Calendar

22.10. Desafio: calculando datas com Calendar

23. Date/Time API: mais nova e moderna

23.1. Introdução à Date and Time API e ao padrão ISO-8601

23.2. Instant: representando um momento na linha do tempo

23.3. LocalDate: representando apenas a data

23.4. LocalTime: representando apenas o horário

23.5. LocalDateTime: representando data e hora
23.6. Outras formas de obter instâncias de LocalDate, LocalTime e LocalDateTime
23.7. Formatando data/hora da nova API
23.8. Convertendo de String para objetos temporais
23.9. Desafio: LocalDate, LocalTime e LocalDateTime
23.10. Obtendo campos de objetos temporais e a enum ChronoField
23.11. Alterando campos de objetos temporais com métodos with
23.12. Adicionando e subtraindo objetos temporais com métodos de conveniência
23.13. Calculando objetos temporais com Chrono Unit
23.14. Desafio: calculadora de parcelas
23.15. Representando períodos com a classe Period
23.16. Calculando períodos de objetos temporais
23.17. Representando durações com a classe Duration
23.18. Calculando durações de objetos temporais
23.19. Desafio: calculando período
23.20. DayOfWeek: representando o dia da semana
23.21. Year: representando o ano
23.22. Month: representando o mês
23.23. Month Day: representando o dia do mês
23.24. Year Month: representando o mês do ano
23.25. Alterando campos de objetos temporais com TemporalAdjusters
23.26. Comparando objetos temporais
23.27. Desafio: TemporalAdjuster

23.28. Identificando fusos com ZoneId e ZoneOffset
23.29. Instanciando objetos temporais em um fuso horário específico
23.30. ZonedDateTime: data/hora com fuso horário
23.31. Calculando e convertendo de/para ZonedDateTime
23.32. Offset DateTime em Offset Time: data e hora com deslocamento do UTC
23.33. Desafio: trabalhando com fuso horário

24. Classes aninhadas

24.1. Introdução às classes aninhadas
24.2. Classes aninhadas estáticas
24.3. Modificadores de acesso de classes aninhadas
24.4. Enum como membro estático de uma classe
24.5. Classes aninhadas não-estáticas
24.6. Shadowing em classes aninhadas
24.7. Classes locais
24.8. Classes anônimas
24.9. Desafio: classes aninhadas

25. Expressões Lambda e Method Reference

25.1. Introdução ao módulo
25.2. Implementando Expressões Lambda
25.3. Entendendo as interfaces funcionais
25.4. Usando a interface @FunctionalInterface
25.5. Boas práticas: prefira lambdas a classes anônimas
25.6. Boas práticas: torne as lambdas mais concisas

- 25.7. Implementando Comparator com lambda
- 25.8. Boas práticas: prefira interfaces funcionais padrão
- 25.9. As 4 principais interfaces funcionais
- 25.10. Interface funcional Predicate: removendo elementos de coleções
- 25.11. Interface funcional Consumer: iterando em coleções com forEach
- 25.12. Interface funcional Function: ordenando lista com Comparator.comparing
- 25.13. Usando Method References
- 25.14. Referenciando métodos de uma instância particular
- 25.15. Referenciando métodos estáticos
- 25.16. Referenciando construtores
- 25.17. Desafio: expressões lambda e method reference

26. Optional

- 26.1. O jeito tradicional de evitar NPE
- 26.2. Evoluindo seu código com Optional
- 26.3. Testando valor do Optional com isPresent
- 26.4. Obtendo valor e lançando exceções com or Else Throw
- 26.5. Obtendo valor alternativo com orElse e orElseGet
- 26.6. Obtendo e testando valor com ifPresent e ifPresentOrElse
- 26.7. Testando e filtrando valor com Predicate
- 26.8. Aplicando transformações com map
- 26.9. Aplicando transformações com flatMap
- 26.10. Tipos especiais de Optional para tipos primitivos

- 26.11. Boas práticas ao usar Optional
- 26.12. Desafio: Optional

27. Streams API

- 27.1. Introdução à Streams API e operações básicas
- 27.2. Operação intermediária: Stream.filter
- 27.3. Operação terminal: Stream.forEach
- 27.4. Criando o pipeline com encadeamento de operações
- 27.5. Executando ações intermediárias com o método Stream.peek
- 27.6. Operações terminais de curto-circuito: findFirst e findAny
- 27.7. Testando predicados com Stream.anyMatch, Stream.allMatch e Stream.noneMatch
- 27.8. Ordenando elementos de Streams
- 27.9. Entendendo o que é uma operação intermediária com estado (stateful)
- 27.10. Aplicando transformações com Stream.map
- 27.11. Obtendo um Stream de elementos distintos
- 27.12. Achatando um Stream com Stream.flatMap
- 27.13. Usando as especializações de Stream para tipos primitivos
- 27.14. Entendendo as operações de redução com Stream.reduce
- 27.15. Reduzindo em BigDecimal e usando a função de combinação
- 27.16. Operações de redução que retornam Optional
- 27.17. Operações de redução especiais: sum, average e count
- 27.18. Operações de redução especiais: min e max
- 27.19. Coletando elementos do Stream em lista com Stream.collect
- 27.20. Usando coletores padrão da classe Collectors

- 27.21. Usando coletores de listas não-modificáveis
- 27.22. Coletando elementos do Stream em mapas
- 27.23. Gerando mapas agrupados com `Collectors.groupingBy`
- 27.24. Gerando mapas agrupados com valores agregados
- 27.25. Gerando mapas particionados com `Collectors.partitioningBy`
- 27.26. Outras formas de obter instâncias de Stream
- 27.27. Métodos `Objects.isNull` e `Objects.nonNull`
- 27.28. Boas práticas: prefira funções em streams sem efeito colateral
- 27.29. Desafio: Streams

28. Manipulando arquivos com a API clássica de I/O

- 28.1. Introdução à API clássica de I/O
- 28.2. Instanciando e criando arquivos e pastas com a classe `File`
- 28.3. Obtendo o caminho absoluto e canônico de `File`
- 28.4. Excluindo, renomeando e movendo arquivos e pastas
- 28.5. Obtendo informações de arquivos e diretórios
- 28.6. Listando arquivos e diretórios
- 28.7. Entendendo I/O streams e Byte-oriented streams
- 28.8. Lendo arquivos com `FileInputStream`
- 28.9. Boa prática: tratando `IOException` com `try-with-resources`
- 28.10. Escrevendo arquivos com `FileOutputStream`
- 28.11. Conhecendo Character-oriented streams
- 28.12. Lendo arquivos de texto com `FileReader`
- 28.13. Escrevendo arquivos de texto com `FileWriter`
- 28.14. Lendo arquivos texto de forma otimizada com `BufferedReader`

- 28.15. Escrevendo arquivos texto de forma otimizada com `BufferedWriter`
- 28.16. Reconhecendo a API de I/O em `System.in` e `Scanner`
- 28.17. Reconhecendo a API de I/O em `System.out` e a classe `PrintStream`
- 28.18. Desafio: API clássica de I/O

29. Manipulando arquivos com NIO.2

- 29.1. Introdução ao NIO é NIO.2
- 29.2. Representando arquivos e pastas com a classe `Path`
- 29.3. Trabalhando com caminhos absolutos e relativos
- 29.4. Operações básicas com a classe `Files`
- 29.5. Copiando arquivos e diretórios
- 29.6. Movendo arquivos e diretórios
- 29.7. Excluindo arquivos e diretórios
- 29.8. Realizando operações com `Files.walkFileTree`
- 29.9. Obtendo informações de arquivos e diretórios
- 29.10. Listando conteúdo de diretórios
- 29.11. Pesquisando arquivos em uma pasta e subpastas
- 29.12. Entendendo os buffers e usando `ByteBuffer`
- 29.13. Usando `CharBuffer`
- 29.14. Decodificando `ByteBuffer` em `CharBuffer`
- 29.15. Lendo arquivos com `ByteChannel`
- 29.16. Lendo arquivos com buffers menores
- 29.17. Escrevendo arquivos com `ByteChannel`
- 29.18. Usando a API de I/O clássica com implementações da NIO
- 29.19. Simplificando a leitura de arquivos com a classe `Files`

29.20. Simplificando a escrita de arquivos com a classe Files

29.21. Desafio: NIO.2

30. Serialização de objetos

30.1. Introdução à serialização de objetos

30.2. Tornando classes serializáveis com a interface Serializable

30.3. Serializando objetos com ObjectOutputStream

30.4. Desserializar objetos com ObjectInputStream

30.5. Ignorando propriedades com transient

30.6. Entendendo e gerando serialVersionUID

30.7. Boas práticas de serialização e serialVersionUID

30.8. Desafio: serialização de objetos

31. Arquivos JAR e Apache Maven

31.1. O que são os arquivos JAR

31.2. Gerando arquivos JAR como bibliotecas

31.3. Importando arquivos JAR no projeto

31.4. Usando bibliotecas parceiras

31.5. Gerando arquivos JAR executável

31.6. O que é Apache Maven?

31.7. Instalando o Apache Maven no Windows

31.8. Instalando o Apache Maven no macOS e Linux

31.9. Criando um projeto Maven com IntelliJ IDEA

31.10. Arquivo pom.xml e Maven Coordinates

31.11. Entendendo o Standard Directory Layout

31.12. Compilando e empacotando com Maven

31.13. Conhecendo os tipos de repositórios Maven

31.14. Instalando e adicionando dependências com Maven

31.15. Usando dependência do Maven Central Repository

31.16. Entendendo as dependências transitivas

31.17. Entendendo os plugins do Maven e o Super POM

31.18. Gerando Fat JAR com Maven Assembly Plugin

32. Java Logging API, Logback e SLF4J

32.1. Por que fazer logging?

32.2. Principais frameworks de logging

32.3. Usando o Java Logging API (JUL)

32.4. Conhecendo os níveis de log do JUL

32.5. Registrando um log de exceção com JUL

32.6. Criando logging.properties e configurando nível de log

32.7. Salvando logs em arquivos com FileHandler do JUL

32.8. Usando a Java Logging API com SLF4J

32.9. Usando o Logback com SLF4J

32.10. Conhecendo os níveis de log do Logback e SLF4J

32.11. Configurando a saída de logs com logback.xml

32.12. Customizando mais o padrão de layout do Encoder

32.13. Salvando logs em arquivos com FileAppender do Logback

32.14. Desafio: Logback e SLF4J

33. Banco de dados e JDBC

- 33.1. Introdução ao MySQL
- 33.2. Instalando o MySQL Server
- 33.3. Instalando o MySQL Workbench
- 33.4. Criando uma tabela no banco de dados
- 33.5. Introdução ao JDBC
- 33.6. Adicionando driver JDBC ao projeto e criando uma conexão
- 33.7. Executando consultas SQL com Statement
- 33.8. Realizando consultas SQL com a cláusula where
- 33.9. Executando consultas SQL com PreparedStatement
- 33.10. Executando comandos que alteram dados (DML)
- 33.11. Obtendo código de autoincremento gerado
- 33.12. Gerenciando transações com o banco de dados

34. Padrão de projeto: Repository

- 34.1. Introdução ao design de software
- 34.2. Criando entidade e camada de serviço
- 34.3. Conhecendo o pattern Repository
- 34.4. Implementando uma classe Repository
- 34.5. Implementando uma consulta no repositório
- 34.6. Desacoplando a conexão com o banco de dados
- 34.7. Implementando uma fábrica de Repository
- 34.8. Desacoplando a implementação de Repository
- 34.9. Alternando a implementação do Repository
- 34.10. Abstraindo a fábrica de repositórios
- 34.11. Criando um arquivo de configuração (properties)

35. Reflection API, anotações e classes seladas

- 35.1. Conhecendo o projeto de gerador de CSV
- 35.2. Lendo propriedades dinamicamente com Reflection API
- 35.3. Invocando a leitura de valores de propriedades
- 35.4. Introdução às anotações do Java
- 35.5. Criando uma anotação customizada
- 35.6. Processando anotações com Reflection API
- 35.7. Adicionando e processando atributos na anotação
- 35.8. Classes seladas
- 35.9. Conclusão e próximos passos

Combo de formações

- ✓ Acesso vitalício
- ✓ Suporte estendido por 24 meses
- ✓ Certificado de conclusão
- ✓ Comunidade no Discord
- ✓ Acesso ao Especialista Java
- ✓ Acesso ao Especialista Spring REST
- ✓ Acesso ao Especialista JPA

~~R\$6.140~~

OFERTA LENDÁRIA

Apenas
12x R\$279

ou R\$2.880 à vista

algaworks
BLACK

- ✓ Assinatura anual
- ✓ Suporte ilimitado
- ✓ Certificado de conclusão
- ✓ Comunidade no Discord
- ✓ Acesso ao Especialista Java
- ✓ Acesso ao Especialista Spring REST
- ✓ Acesso ao Especialista JPA
- ✓ Acesso a 22 cursos da plataforma
- ✓ Atualizações gratuitas
- ✓ Acesso a futuras regravações de cursos
- ✓ Acesso a novos cursos lançados selecionados

~~R\$252 por mês~~

OFERTA LENDÁRIA

R\$182 por mês
no plano anual

ou de ~~R\$2.580~~ por R\$1.880 à vista

ESPECIALISTA
JAVA

- ✓ Acesso vitalício
- ✓ Suporte por 12 meses
- ✓ Certificado de conclusão
- ✓ Comunidade no Discord
- ✓ Acesso ao Especialista Java
- ✗ Acesso ao Especialista Spring REST
- ✗ Acesso ao Especialista JPA

~~R\$1.980~~

OFERTA LENDÁRIA

Apenas
12x R\$163

ou R\$1.680 à vista

ESPECIALISTA
SPRING REST

- ✓ Acesso vitalício
- ✓ Suporte por 12 meses
- ✓ Certificado de conclusão
- ✓ Comunidade no Discord
- ✗ Acesso ao Especialista Java
- ✓ Acesso ao Especialista Spring REST
- ✗ Acesso ao Especialista JPA

~~R\$2.580~~

OFERTA LENDÁRIA

Apenas
12x R\$210

ou R\$2.180 à vista

ESPECIALISTA
JPA

- ✓ Acesso vitalício
- ✓ Suporte por 12 meses
- ✓ Certificado de conclusão
- ✓ Comunidade no Discord
- ✗ Acesso ao Especialista Java
- ✗ Acesso ao Especialista Spring REST
- ✓ Acesso ao Especialista JPA

~~R\$1.580~~

OFERTA LENDÁRIA

Apenas
12x R\$130

ou R\$1.340 à vista