Caderno de Exercícios: Programação Orientada a Objetos em Java e JDBC

Apresentação

Este caderno de exercícios foi elaborado para auxiliar no estudo e aprofundamento dos conceitos fundamentais de Programação Orientada a Objetos (POO) em Java e JDBC (Java Database Connectivity). As questões discursivas aqui apresentadas abordam temas essenciais como os pilares da POO, tratamento de exceções, conexão com bancos de dados PostgreSQL e MySQL, e bibliotecas de persistência.

Cada questão contém uma introdução contextualizada sobre o tema, seguida de itens e subitens que exploram diferentes aspectos do assunto. Recomenda-se que as respostas sejam elaboradas de forma completa, demonstrando compreensão dos conceitos e suas aplicações práticas.

Bom estudo!

Questão 1: Tratamento de Exceções em Java

Contextualização

O tratamento de exceções é um mecanismo fundamental em Java que permite lidar com situações anormais ou erros que podem ocorrer durante a execução de um programa. Diferentemente de outras linguagens que utilizam códigos de erro, Java implementa um sistema robusto baseado em objetos para representar e gerenciar exceções. Este sistema permite separar o código normal do código de tratamento de erros, tornando os programas mais legíveis, organizados e confiáveis. O mecanismo de exceções em Java é baseado em blocos try-catch-finally, que permitem capturar e tratar diferentes tipos de exceções de maneira estruturada, além de garantir a execução de código de limpeza independentemente da ocorrência de erros.

Questão

a) Explique detalhadamente o mecanismo de tratamento de exceções em Java, descrevendo a sintaxe e o funcionamento dos blocos try, catch, finally e throw. Discuta

também a diferença entre exceções verificadas (checked) e não verificadas (unchecked), fornecendo exemplos de cada tipo.

b) Analise o seguinte trecho de código e explique o que aconteceria em cada um dos cenários descritos:

```
public void processarArquivo(String caminho) {
    FileInputStream arquivo = null;
    try {
        arquivo = new FileInputStream(caminho);
        // Processamento do arquivo
        int dado = arquivo.read();
        if (dado == -1) {
            throw new IOException("Arquivo vazio");
        }
        return;
    } catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado: " +
e.getMessage());
    } catch (IOException e) {
        System.out.println("Erro de I/O: " + e.getMessage());
    } finally {
        if (arquivo != null) {
            try {
                arquivo.close();
            } catch (IOException e) {
                System.out.println("Erro ao fechar arquivo: " +
e.getMessage());
            }
        }
        System.out.println("Execução do bloco finally");
    }
    System.out.println("Fim do método");
}
```

Cenários: 1. O arquivo existe e contém dados 2. O arquivo existe, mas está vazio 3. O arquivo não existe 4. Ocorre um erro ao fechar o arquivo

c) Crie uma hierarquia de exceções personalizadas para um sistema bancário que inclua pelo menos três tipos específicos de exceções (por exemplo, SaldoInsuficienteException, ContaBloqueadaException, ValorInvalidoException). Implemente um exemplo de código que demonstre o lançamento e tratamento dessas exceções em operações bancárias, utilizando múltiplos blocos catch e o mecanismo de try-with-resources introduzido no Java 7.

Questão 2: JDBC e Conexão com Bancos de Dados

Contextualização

JDBC (Java Database Connectivity) é uma API padrão do Java que permite a conexão e interação com diversos sistemas de gerenciamento de bancos de dados (SGBDs) como PostgreSQL, MySQL, Oracle, entre outros. Ela fornece um conjunto de classes e interfaces que permitem executar operações SQL, processar resultados e gerenciar transações de forma independente do banco de dados utilizado. O JDBC atua como uma camada de abstração entre a aplicação Java e o banco de dados, permitindo que os desenvolvedores escrevam código que pode funcionar com diferentes SGBDs com alterações mínimas. Essa flexibilidade é possível graças aos drivers JDBC específicos para cada banco de dados, que implementam as interfaces padrão definidas pela API.

Questão

a) Explique o funcionamento da API JDBC, detalhando seus principais componentes (Driver, Connection, Statement, PreparedStatement, ResultSet) e o fluxo típico de operações para executar consultas e atualizações em um banco de dados. Compare as diferenças entre Statement, PreparedStatement e CallableStatement, destacando as vantagens e desvantagens de cada um.

b) Implemente um código completo em Java que demonstre a conexão com um banco de dados PostgreSQL e outro com MySQL, realizando as seguintes operações: 1. Estabelecer conexão com o banco de dados 2. Criar uma tabela de produtos (com campos como id, nome, preço, quantidade) 3. Inserir múltiplos registros utilizando PreparedStatement 4. Realizar uma consulta com filtros e ordenação 5. Atualizar registros 6. Excluir registros 7. Implementar tratamento adequado de exceções 8. Garantir o fechamento correto dos recursos

c) Discuta as melhores práticas de segurança e desempenho ao utilizar JDBC em aplicações Java, abordando os seguintes aspectos: 1. Prevenção de injeção de SQL 2. Gerenciamento eficiente de conexões (connection pooling) 3. Tratamento de transações 4. Manipulação de grandes conjuntos de dados (streaming de resultados) 5. Estratégias para lidar com diferentes dialetos SQL entre PostgreSQL e MySQL

Questão 3: Pilares da Programação Orientada a Objetos

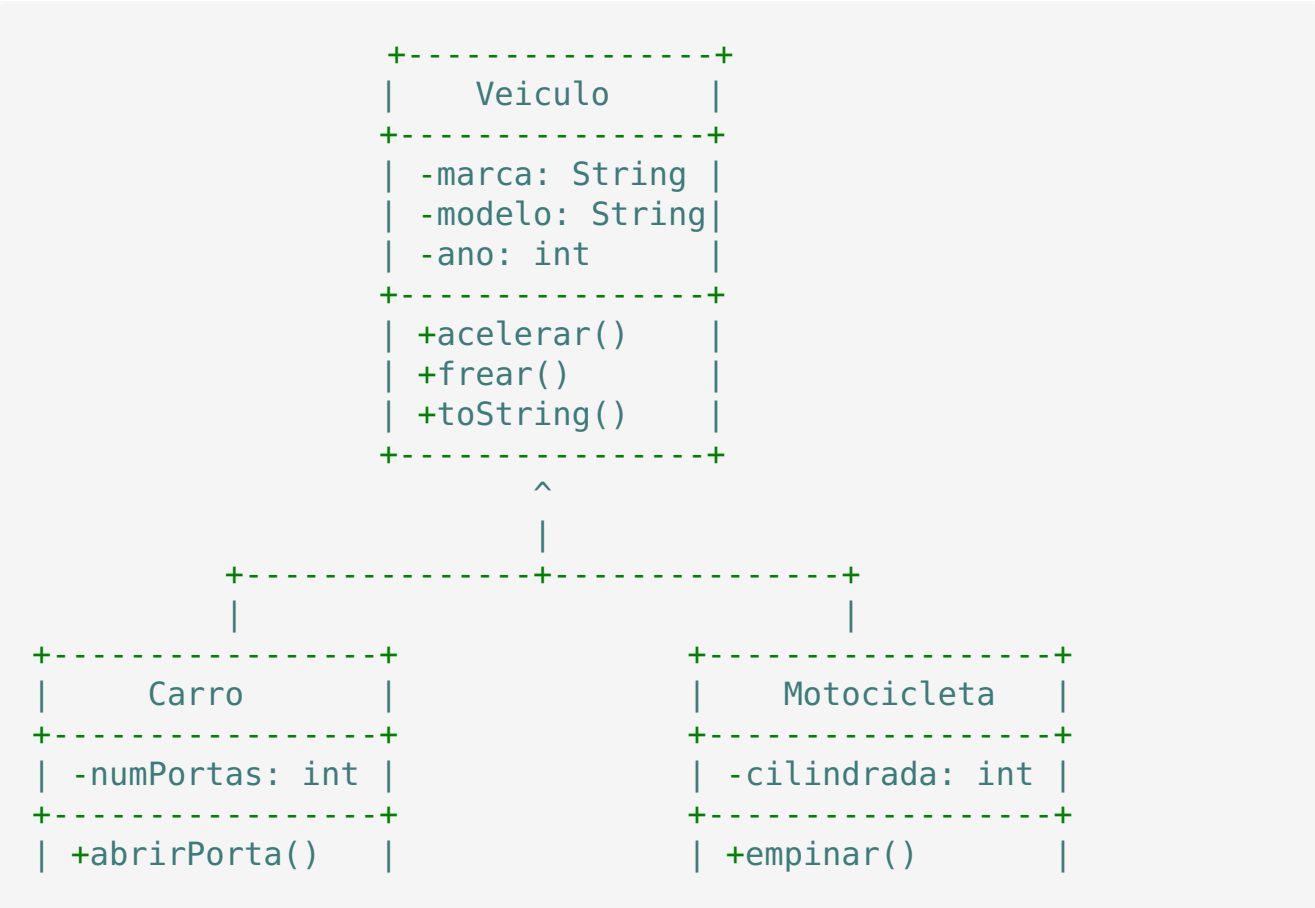
Contextualização

A Programação Orientada a Objetos (POO) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos (atributos) e código na forma de procedimentos (métodos). A POO foi desenvolvida para facilitar o desenvolvimento de software, tornando-o mais modular, reutilizável e fácil de manter. Java é uma linguagem fortemente orientada a objetos que implementa os quatro pilares fundamentais da POO: encapsulamento, herança, polimorfismo e abstração. Esses conceitos permitem aos desenvolvedores criar sistemas complexos a partir de componentes simples e reutilizáveis, facilitando a manutenção e evolução do software ao longo do tempo.

Questão

a) Defina e explique detalhadamente cada um dos quatro pilares da Programação Orientada a Objetos (encapsulamento, herança, polimorfismo e abstração), fornecendo exemplos práticos em Java para cada conceito. Discuta como esses pilares se complementam e como contribuem para o desenvolvimento de software de qualidade.

b) Analise o seguinte diagrama de classes UML e implemente-o em Java, demonstrando os conceitos de herança, polimorfismo, encapsulamento e abstração:



```
| +acelerar()     |                | +acelerar()      |
| +toString()     |                | +toString()      |
+-----------------+                +------------------+
```

c) Explique o conceito de polimorfismo em Java, distinguindo entre polimorfismo de sobrecarga (overloading) e polimorfismo de sobreposição (overriding). Crie um exemplo prático que demonstre ambos os tipos de polimorfismo em um sistema de processamento de pagamentos que aceita diferentes formas de pagamento (cartão de crédito, boleto, transferência bancária). Discuta como o polimorfismo contribui para a flexibilidade e extensibilidade desse sistema.

Questão 4: Classes Abstratas e Interfaces em Java

Contextualização

Classes abstratas e interfaces são mecanismos fundamentais em Java para implementar abstração e polimorfismo. Elas permitem definir contratos que as classes concretas devem seguir, estabelecendo um conjunto de métodos que devem ser implementados. Enquanto as classes abstratas podem conter implementações parciais e estado (atributos), as interfaces tradicionalmente definem apenas assinaturas de métodos (embora a partir do Java 8 possam incluir métodos default e estáticos). A escolha entre usar uma classe abstrata ou uma interface depende do contexto e dos requisitos específicos do design do sistema. Compreender as diferenças, vantagens e limitações de cada abordagem é essencial para criar hierarquias de classes eficientes e flexíveis.

Questão

a) Compare classes abstratas e interfaces em Java, explicando suas diferenças, vantagens e desvantagens. Discuta as mudanças introduzidas nas interfaces a partir do Java 8 (métodos default e estáticos) e do Java 9 (métodos privados) e como essas mudanças afetaram o design de aplicações Java. Em quais situações é mais apropriado usar uma classe abstrata e em quais é preferível usar uma interface?

b) Implemente um sistema de formas geométricas em Java que demonstre o uso adequado de classes abstratas e interfaces. O sistema deve incluir: 1. Uma classe abstrata `FormaGeometrica` com métodos abstratos para calcular área e perímetro 2. Classes concretas como `Circulo`, `Retangulo`, `Triangulo` que estendem `FormaGeometrica` 3. Uma interface `Desenhavel` com métodos para desenhar e redimensionar 4. Uma interface `Rotacionavel` com métodos para rotacionar 5.

Implementação apropriada das interfaces nas classes concretas 6. Uma classe de teste que demonstre polimorfismo usando tanto a classe abstrata quanto as interfaces

c) Discuta o problema do "diamante" em herança múltipla e como Java resolve esse problema através de interfaces. Crie um exemplo que demonstre como as interfaces e os métodos default podem levar a ambiguidades semelhantes ao problema do diamante e como resolvê-las em Java. Explique também o conceito de "programação para interfaces, não para implementações" e como ele se relaciona com os princípios SOLID de design orientado a objetos.

Questão 5: Bibliotecas de Persistência em Java

Contextualização

A persistência de dados é um aspecto crucial no desenvolvimento de aplicações empresariais em Java. Embora o JDBC forneça uma API de baixo nível para interagir com bancos de dados, ele frequentemente exige código repetitivo e propenso a erros. Para resolver esse problema, surgiram diversas bibliotecas e frameworks de persistência que abstraem a complexidade do JDBC e oferecem recursos adicionais como mapeamento objeto-relacional (ORM), gerenciamento de transações declarativo e consultas orientadas a objetos. Entre as principais soluções estão JPA (Java Persistence API), Hibernate, MyBatis, Spring Data e outras tecnologias que simplificam o acesso a dados e melhoram a produtividade dos desenvolvedores.

Questão

a) Compare as seguintes tecnologias de persistência em Java: JDBC, JPA/Hibernate, MyBatis e Spring Data. Para cada uma, explique seu funcionamento básico, arquitetura, vantagens e desvantagens. Discuta também os cenários em que cada tecnologia seria mais apropriada, considerando fatores como complexidade do projeto, desempenho, curva de aprendizado e manutenibilidade.

b) Implemente um exemplo de persistência de dados utilizando JPA/Hibernate para um sistema de gerenciamento de biblioteca. O sistema deve incluir: 1. Entidades JPA para `Livro`, `Autor`, `Categoria` e `Emprestimo` com os relacionamentos apropriados (um-para-um, um-para-muitos, muitos-para-muitos) 2. Configuração do persistence.xml para PostgreSQL 3. Implementação de um DAO (Data Access Object) genérico e DAOs específicos para cada entidade 4. Demonstração de operações CRUD (Create, Read, Update, Delete) 5. Implementação de consultas JPQL e Criteria API 6. Gerenciamento adequado de transações

c) Discuta as melhores práticas para o uso de JPA/Hibernate em aplicações empresariais, abordando os seguintes aspectos: 1. Estratégias de mapeamento objeto-relacional 2. Gerenciamento de cache de primeiro e segundo nível 3. Carregamento lazy vs. eager e o problema do N+1 4. Otimização de consultas e considerações de desempenho 5. Integração com outras tecnologias como Spring Framework 6. Migração de esquema de banco de dados (usando ferramentas como Flyway ou Liquibase)

Questão 6: Padrões de Projeto em Java

Contextualização

Padrões de projeto (Design Patterns) são soluções reutilizáveis para problemas comuns encontrados no design de software. Eles representam as melhores práticas utilizadas por desenvolvedores experientes e fornecem um vocabulário comum para discutir arquiteturas de software. Em Java, os padrões de projeto são amplamente utilizados tanto na biblioteca padrão quanto em frameworks populares. Eles são categorizados principalmente em três grupos: padrões criacionais (relacionados à criação de objetos), padrões estruturais (relacionados à composição de classes e objetos) e padrões comportamentais (relacionados à interação entre objetos). O conhecimento desses padrões é essencial para desenvolver software bem estruturado, flexível e manutenível.

Questão

a) Explique o conceito de padrões de projeto e sua importância no desenvolvimento de software orientado a objetos. Descreva detalhadamente três padrões de projeto de cada categoria (criacional, estrutural e comportamental), fornecendo exemplos de implementação em Java e discutindo situações em que cada um seria aplicável.

b) Implemente um sistema de processamento de pedidos em Java que utilize os seguintes padrões de projeto: 1. Factory Method ou Abstract Factory para criar diferentes tipos de produtos 2. Decorator para adicionar funcionalidades extras aos produtos (como garantia estendida, embalagem especial) 3. Observer para notificar diferentes partes do sistema quando um pedido é criado ou seu status é alterado 4. Strategy para implementar diferentes estratégias de cálculo de frete e desconto 5. Singleton para gerenciar uma conexão com o banco de dados

c) Analise criticamente o uso de padrões de projeto em Java, discutindo: 1. Como identificar quando um padrão de projeto é realmente necessário 2. Os riscos de aplicar padrões de projeto de forma excessiva ou inadequada (overengineering) 3. A relação entre padrões de projeto e princípios SOLID 4. Como os recursos modernos do Java

(lambdas, streams, módulos) afetaram a implementação de padrões de projeto tradicionais 5. Padrões de projeto específicos para aplicações web e empresariais em Java

---

Questão 7: Programação Concorrente em Java

Contextualização

A programação concorrente permite que múltiplas tarefas sejam executadas simultaneamente, aproveitando melhor os recursos computacionais modernos, especialmente em sistemas multicore. Java oferece um robusto suporte à concorrência desde suas primeiras versões, com a API de threads básica, e evoluiu significativamente com a introdução do pacote java.util.concurrent no Java 5. A programação concorrente traz benefícios como melhor desempenho e responsividade, mas também introduz desafios como condições de corrida, deadlocks e starvation. Dominar os conceitos e ferramentas de concorrência em Java é essencial para desenvolver aplicações escaláveis e eficientes em ambientes multithread.

Questão

a) Explique os conceitos fundamentais de programação concorrente em Java, incluindo threads, sincronização, monitores, variáveis voláteis e o modelo de memória Java. Discuta os problemas comuns em programação concorrente (condições de corrida, deadlocks, livelocks, starvation) e como evitá-los. Compare a API de threads tradicional (Thread, Runnable, synchronized) com as abstrações de alto nível do pacote java.util.concurrent.

b) Implemente um sistema de processamento paralelo de arquivos em Java que: 1. Leia múltiplos arquivos simultaneamente usando um pool de threads 2. Processe o conteúdo de cada arquivo (por exemplo, contagem de palavras, busca de padrões) 3. Combine os resultados parciais em um resultado final 4. Utilize classes do pacote java.util.concurrent como ExecutorService, Future, CountDownLatch, ConcurrentHashMap 5. Implemente mecanismos adequados de tratamento de exceções em ambiente multithread 6. Demonstre o uso de recursos do Java 8+ como CompletableFuture para operações assíncronas

c) Discuta as melhores práticas para desenvolvimento de aplicações concorrentes em Java, abordando: 1. Estratégias para teste de código concorrente 2. Ferramentas para detecção de problemas de concorrência 3. Padrões de concorrência (como produtor-consumidor, leitores-escritores, fork-join) 4. Considerações de desempenho e

escalabilidade 5. Programação reativa com bibliotecas como RxJava ou Project Reactor 6. Concorrência em ambientes distribuídos (microserviços, computação em nuvem)

Questão 8: Coleções e Generics em Java

Contextualização

O framework de coleções Java (Java Collections Framework) fornece uma arquitetura unificada para representar e manipular grupos de objetos. Ele inclui interfaces, implementações e algoritmos que operam sobre coleções, permitindo armazenar, recuperar e processar dados de forma eficiente. Combinado com o sistema de tipos genéricos (Generics), introduzido no Java 5, o framework de coleções oferece segurança de tipo em tempo de compilação e elimina a necessidade de casting explícito. Essas tecnologias são fundamentais para o desenvolvimento de aplicações Java robustas e eficientes, pois permitem criar estruturas de dados reutilizáveis e algoritmos que operam sobre diferentes tipos de objetos.

Questão

a) Explique a arquitetura do Java Collections Framework, descrevendo suas principais interfaces (Collection, List, Set, Map, Queue) e implementações (ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap). Compare o desempenho das diferentes implementações em termos de operações comuns (inserção, remoção, busca, iteração) e discuta critérios para escolher a implementação mais adequada para diferentes cenários.

b) Demonstre o uso de Generics em Java através de exemplos práticos que incluam: 1. Definição de classes e interfaces genéricas 2. Métodos genéricos 3. Wildcards (? extends T, ? super T) 4. Restrições de tipo (type bounds) 5. Erasure de tipo e suas implicações 6. Implementação de uma estrutura de dados genérica personalizada (como uma árvore binária de busca)

c) Implemente um sistema de análise de dados em Java que utilize recursos avançados de coleções e Generics para: 1. Carregar dados de diferentes fontes (arquivos CSV, banco de dados) 2. Filtrar, mapear e reduzir coleções usando a Stream API do Java 8+ 3. Agrupar e sumarizar dados usando Collectors 4. Implementar algoritmos de ordenação personalizados com Comparator 5. Utilizar coleções imutáveis e thread-safe quando apropriado 6. Demonstrar o uso de Optional para evitar NullPointerException

---

Questão 9: Testes e Qualidade de Código em Java

Contextualização

Testes automatizados e práticas de qualidade de código são essenciais para garantir a confiabilidade, manutenibilidade e evolução contínua de aplicações Java. Metodologias como Test-Driven Development (TDD), ferramentas de teste como JUnit e Mockito, e análise estática de código com ferramentas como SonarQube e CheckStyle, ajudam a identificar problemas precocemente no ciclo de desenvolvimento, reduzindo o custo de correção de bugs e melhorando a qualidade geral do software. Além disso, práticas como integração contínua e entrega contínua (CI/CD) permitem validar automaticamente as mudanças no código e entregar software de forma mais rápida e confiável.

Questão

a) Explique os diferentes tipos de testes em aplicações Java (unitários, integração, sistema, aceitação) e as ferramentas comumente utilizadas para cada tipo. Discuta os princípios do Test-Driven Development (TDD) e como implementá-lo em projetos Java. Explique também o conceito de cobertura de código e sua importância na avaliação da qualidade dos testes.

b) Implemente testes unitários completos para um DAO (Data Access Object) que utiliza JDBC para acessar um banco de dados. Os testes devem: 1. Utilizar JUnit 5 como framework de teste 2. Implementar mocks com Mockito para simular o comportamento do banco de dados 3. Testar cenários de sucesso e falha para operações CRUD 4. Utilizar assertions adequadas para verificar os resultados 5. Implementar fixtures e setup/teardown apropriados 6. Demonstrar o uso de testes parametrizados e testes dinâmicos

c) Discuta as melhores práticas para garantir a qualidade de código em projetos Java, abordando: 1. Ferramentas de análise estática de código (SonarQube, CheckStyle, PMD, FindBugs) 2. Convenções de codificação e sua importância 3. Refatoração segura com suporte de testes automatizados 4. Integração contínua e entrega contínua (CI/CD) para projetos Java 5. Revisão de código e programação em par 6. Métricas de qualidade de código e como interpretá-las

---

Questão 10: Desenvolvimento Web com Java

Contextualização

O desenvolvimento web é uma das áreas mais importantes de aplicação da linguagem Java, com diversas tecnologias e frameworks disponíveis para criar aplicações web robustas, escaláveis e seguras. Desde os primeiros Servlets e JSPs até modernos frameworks como Spring Boot, Jakarta EE (anteriormente Java EE) e Quarkus, o ecossistema Java oferece soluções para diferentes necessidades e escalas de projetos. Essas tecnologias permitem implementar desde simples APIs RESTful até complexos sistemas empresariais distribuídos, integrando-se com bancos de dados, serviços externos e front-ends baseados em JavaScript.

Questão

a) Compare as principais tecnologias e frameworks para desenvolvimento web em Java (Servlets/JSP, Jakarta EE, Spring MVC/Boot, Quarkus, Micronaut), explicando suas características, vantagens e desvantagens. Discuta a evolução do desenvolvimento web em Java, desde as abordagens tradicionais baseadas em servidor até arquiteturas modernas como microserviços e aplicações reativas.

b) Implemente uma API RESTful em Java utilizando Spring Boot para um sistema de gerenciamento de tarefas (to-do list) que: 1. Ofereça endpoints para operações CRUD de tarefas 2. Implemente autenticação e autorização com Spring Security e JWT 3. Utilize Spring Data JPA para persistência em banco de dados 4. Implemente validação de dados de entrada 5. Inclua documentação da API com Swagger/OpenAPI 6. Implemente testes unitários e de integração 7. Utilize boas práticas de tratamento de exceções e respostas HTTP

c) Discuta os desafios e melhores práticas no desenvolvimento de aplicações web modernas com Java, abordando: 1. Arquiteturas de microserviços vs. monolíticas 2. Containerização com Docker e orquestração com Kubernetes 3. Comunicação assíncrona entre serviços (mensageria, eventos) 4. Monitoramento e observabilidade de aplicações 5. Estratégias de implantação (deployment) e DevOps 6. Integração com front-ends modernos baseados em frameworks JavaScript