Aplicações Práticas em Java - Do Básico ao Avançado, Incluindo Inteligência Artificial, Machine Learning, GenAI e IBM ODM

Introdução Java é uma das linguagens de programação mais consagradas e amplamente adotadas em empresas de todo o mundo.

Com uma sintaxe robusta e suporte multiplataforma, Java possibilita a criação de sistemas escaláveis, seguros e de alta performance, adaptando-se a aplicações que vão desde calculadoras simples até sistemas completos de e-commerce e soluções de Inteligência Artificial.

Este artigo apresenta exemplos de aplicações desenvolvidas em três níveis de complexidade: básico, intermediário e avançado, além de incluir seções específicas sobre aplicações em Inteligência Artificial (IA), Machine Learning, Generative AI (GenAI) e IBM Operational Decision Manager (ODM). Vamos explorar como Java pode atender a diferentes demandas no desenvolvimento de software.

1. Aplicações Básicas em Java

Exemplo: Calculadora de Operações Matemáticas

Aplicações de nível básico em Java envolvem conceitos fundamentais como manipulação de variáveis, estruturas de controle e operações básicas. Abaixo, vemos um exemplo clássico: uma calculadora que realiza operações de soma, subtração, multiplicação e divisão. Esse tipo de aplicação é ideal para iniciantes, pois envolve a lógica condicional e manipulação de entradas do usuário.

```
import java.util.Scanner;

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

        System.out.print("Digite o primeiro número: ");
        double num1 = scanner.nextDouble();

        System.out.print("Digite o segundo número: ");
        double num2 = scanner.nextDouble();
```

```
        System.out.print("Escolha a operação (+, -, *, /): ");
        char operacao = scanner.next().charAt(0);

        double resultado;
        switch (operacao) {
            case '+':
                resultado = num1 + num2;
                break;
            case '-':
                resultado = num1 - num2;
                break;
            case '*':
                resultado = num1 * num2;
                break;
            case '/':
                resultado = num1 / num2;
                break;
            default:
                System.out.println("Operação inválida.");
                return;
        }
        System.out.println("Resultado: " + resultado);
    }
}
```

A calculadora permite que o usuário escolha a operação desejada através de uma estrutura switch, que facilita a expansão para incluir outras operações no futuro. Esse projeto simples ajuda novos desenvolvedores a entender a manipulação de dados e a estrutura lógica da linguagem.

2. Aplicações Intermediárias em Java

Exemplo: Gerenciador de Tarefas com Interface Gráfica (GUI)

No nível intermediário, introduzimos a criação de interfaces gráficas e a organização de dados em estruturas mais sofisticadas. Um exemplo interessante é o desenvolvimento de um gerenciador de tarefas usando Java Swing, onde o usuário pode adicionar e visualizar tarefas em uma lista. Este tipo de aplicação já utiliza o paradigma de programação orientada a eventos, permitindo que a aplicação responda dinamicamente às ações do usuário.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class GerenciadorDeTarefas extends JFrame {
    private ArrayList<String> tarefas = new ArrayList<>();
    private DefaultListModel<String> modeloLista = new DefaultListModel<>();
    private JList<String> listaTarefas;
```

```
    public GerenciadorDeTarefas() {
        setTitle("Gerenciador de Tarefas");
        setSize(300, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        listaTarefas = new JList<>(modeloLista);
        add(new JScrollPane(listaTarefas), BorderLayout.CENTER);

        JTextField campoTarefa = new JTextField();
        add(campoTarefa, BorderLayout.NORTH);

        JButton botaoAdicionar = new JButton("Adicionar Tarefa");
        botaoAdicionar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String tarefa = campoTarefa.getText();
                if (!tarefa.isEmpty()) {
                    tarefas.add(tarefa);
                    modeloLista.addElement(tarefa);
                    campoTarefa.setText("");
                }
            }
        });
        add(botaoAdicionar, BorderLayout.SOUTH);
    }

    public static void main(String[] args) {
        GerenciadorDeTarefas app = new GerenciadorDeTarefas();
        app.setVisible(true);
    }
}
```

O gerenciador de tarefas permite que os usuários adicionem tarefas e visualizem a lista. A interface gráfica é desenvolvida com o uso de componentes Swing, que são amplamente utilizados para criar GUIs desktop em Java. A aplicação ilustra a manipulação de listas e a interação com o usuário em um nível mais complexo que o exemplo básico.

3. Aplicações Avançadas em Java

Exemplo: Sistema de E-commerce com Banco de Dados e Spring Boot

Para aplicações avançadas, como um sistema de e-commerce, Java oferece poderosas ferramentas e frameworks, como o Spring Boot e Hibernate, que facilitam o desenvolvimento de APIs REST e a integração com bancos de dados. Um sistema de e-commerce inclui funcionalidades de CRUD (criação, leitura, atualização e exclusão) para gerenciamento de produtos e integração com front-ends via API.

Estrutura do Projeto:

- **Banco de Dados**: MySQL, utilizado para armazenar dados dos produtos e usuários.
- **Backend**: Spring Boot, utilizado para criação das APIs REST.
- **Entidade Produto**: Representa os produtos no sistema.

3.1 Entidade Produto

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private double preco;

    // Construtores, Getters e Setters
}
```

3.2 Interface ProdutoRepositor

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProdutoRepository extends JpaRepository<Produto, Long> {
}
```

3.3 Controlador ProdutoController

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/produtos")
public class ProdutoController {

    @Autowired
    private ProdutoRepository produtoRepository;

    @GetMapping
    public List<Produto> listarProdutos() {
        return produtoRepository.findAll();
    }

    @PostMapping
    public Produto adicionarProduto(@RequestBody Produto produto) {
        return produtoRepository.save(produto);
    }

    @DeleteMapping("/{id}")
    public void deletarProduto(@PathVariable Long id) {
```

```
        produtoRepository.deleteById(id);
    }

M}
```

O Spring Boot simplifica a criação de um backend robusto. Com o uso de @RestController, @GetMapping, @PostMapping, e @DeleteMapping, criamos uma API RESTful que permite realizar operações em um banco de dados MySQL. A aplicação modular possibilita a expansão para integração com front-end e inclusão de funcionalidades avançadas, como autenticação de usuários.

4. Aplicações em Inteligência Artificial (IA)

Java é frequentemente utilizado em projetos de IA devido à sua robustez e eficiência. Um exemplo prático é a implementação de sistemas de recomendação, que utilizam algoritmos de aprendizado de máquina para sugerir produtos ou conteúdos aos usuários.

Exemplo: Sistema de Recomendação Simples

A seguir, apresentamos um exemplo de um sistema de recomendação baseado em um modelo simples de filtragem colaborativa.

```
import java.util.HashMap;
import java.util.List;

public class SistemaRecomendacao {
    private HashMap<String, HashMap<String, Integer>> avaliacaoUsuarios;

    public SistemaRecomendacao() {
        avaliacaoUsuarios = new HashMap<>();
    }

    public void adicionarAvaliacao(String usuario, String produto, int avaliacao) {
        avaliacaoUsuarios.putIfAbsent(usuario, new HashMap<>());
        avaliacaoUsuarios.get(usuario).put(produto, avaliacao);
    }

    public List<String> recomendar(String usuario) {
        // Algoritmo simples de recomendação
        // Para fins de exemplo, este método deve ser implementado com lógica de
recomendação real
        return List.of("Produto A", "Produto B", "Produto C");
    }
```

Este sistema permite que os usuários adicionem avaliações de produtos e, em seguida, recomenda produtos com base em uma lógica simples. Na prática, algoritmos mais complexos são utilizados para calcular as recomendações com maior precisão.

5. Aplicações em Machine Learning

O desenvolvimento de modelos de Machine Learning em Java pode ser realizado usando bibliotecas como Weka, Deeplearning4j e Apache Spark. Essas ferramentas oferecem suporte para tarefas de classificação, regressão e clustering.

Exemplo: Classificador de Dados com Weka

Abaixo está um exemplo de um classificador de dados utilizando a biblioteca Weka para prever a classe de um conjunto de dados.

```
import weka.classifiers.Classifier;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class ClassificadorWeka {
    public static void main(String[] args) throws Exception {
        // Carregar dados
        DataSource source = new DataSource("caminho/para/dados.arff");
        Instances dados = source.getDataSet();
        dados.setClassIndex(dados.numAttributes() - 1);

        // Criar classificador
        Classifier classificador = new J48(); // Árvore de decisão J48
        classificador.buildClassifier(dados);

        // Realizar previsões
        // Implementar lógica de previsão com novos dados
    }
}
```

Neste exemplo, a biblioteca Weka é utilizada para carregar um conjunto de dados em formato ARFF e treinar um classificador baseado em uma árvore de decisão.

A versatilidade do Weka permite explorar diversos algoritmos de aprendizado, tornando-se uma ferramenta poderosa para desenvolvedores que desejam integrar Machine Learning em suas aplicações Java.

6. Aplicações em Generative AI (GenAI)

Generative AI refere-se a algoritmos que podem criar novos conteúdos, como imagens, texto e música, em resposta a entradas específicas. Java pode ser usado para integrar modelos de GenAI, utilizando APIs de serviços como OpenAI, que oferecem acesso a modelos de linguagem e geração de conteúdo.

Exemplo: Integração com API de Geração de Texto

A seguir, um exemplo básico de como integrar uma aplicação Java com uma API de Geração de Texto (como OpenAI) para criar conteúdo automaticamente.

```
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class GeradorTexto {
    public static void main(String[] args) throws Exception {
        String apiKey = "SUA_CHAVE_API";
        String prompt = "Crie um resumo sobre Inteligência Artificial.";

        URL url = new URL("https://api.openai.com/v1/engines/davinci/completions");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Authorization", "Bearer " + apiKey);
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setDoOutput(true);

        String jsonInputString = "{ \"prompt\": \"" + prompt + "\", \"max_tokens\": 100 }";
        try (OutputStream os = conn.getOutputStream()) {
            byte[] input = jsonInputString.getBytes("utf-8");
            os.write(input, 0, input.length);
        }

        // Ler a resposta da API
        // Implementar lógica para ler e processar a resposta
    }
}
```

Explicação: O código acima ilustra como enviar uma solicitação a uma API de Geração de Texto. A aplicação pode ser expandida para processar e exibir o conteúdo gerado ao usuário, ilustrando como Java pode ser utilizado em aplicações inovadoras de GenAI.

7. Aplicações com IBM ODM

IBM Operational Decision Manager (ODM) é uma solução que permite a automação de decisões de negócios, utilizando regras de negócio, tabelas de decisão e processos de modelagem. Java é frequentemente utilizado para implementar a lógica de negócios e as integrações necessárias.

Exemplo: Criação de Classe Java XOM e Modelagem de Dados

Abaixo, um exemplo de como criar uma classe Java (XOM - XML Object Model) para ser utilizada no IBM ODM.

7.1 Criação da Classe XOM

```
public class Pedido {
```

```
    private String id;
    private double valor;

    public Pedido(String id, double valor) {
        this.id = id;
        this.valor = valor;
    }

    // Getters e Setters
    public String getId() {
        return id;
    }

    public double getValor() {
        return valor;
    }
}
```

7.2 Modelagem de Dados com Verbalização em Português

No IBM ODM, a modelagem de dados pode ser realizada utilizando uma linguagem de regras que permite definir como os dados serão tratados.

Regra de Exemplo:

Se Pedido.valor > 1000 então
Ação: Aprovar Pedido
Senão
Ação: Rejeitar Pedido

Esta regra simples ilustra como decisões podem ser automatizadas com base em condições específicas.

A verbalização permite que os analistas de negócios entendam facilmente a lógica por trás das regras.

7.3 Tabelas de Decisão

As tabelas de decisão são uma forma visual de representar as regras de negócio e são particularmente úteis em ambientes complexos.

Exemplo de Tabela de Decisão:

Valor do Pedido	Ação
> 1000	Aprovar Pedido
<= 1000	Rejeitar Pedido

As tabelas de decisão organizam as regras de maneira clara e concisa, facilitando a manutenção e a comunicação entre as partes interessadas.

Conclusão

Java se apresenta como uma linguagem versátil, apta para desenvolver aplicações que vão desde soluções simples até sistemas complexos integrados com Inteligência Artificial, Machine Learning, Generative AI e IBM ODM.

Este artigo destacou exemplos práticos e abordagens que facilitam a compreensão do potencial da linguagem na criação de softwares robustos e escaláveis.

A usabilidade das aplicações em Java é vasta e se adapta a diferentes contextos e necessidades.

Desde a criação de pequenas ferramentas, como calculadoras e gerenciadores de tarefas, até sistemas robustos de e-commerce e integrações com tecnologias emergentes como IA e GenAI, Java continua a ser uma escolha popular entre desenvolvedores.

A inclusão de IBM ODM demonstra como Java pode ser utilizado para automação de decisões de negócios, permitindo que as empresas agilizem processos e melhorem a eficiência operacional.

Com a crescente demanda por soluções baseadas em IA e automação, o domínio das aplicações em Java e suas integrações se torna cada vez mais valioso.

Profissionais que se especializam em Java e suas bibliotecas, frameworks e tecnologias relacionadas têm uma vantagem significativa no mercado de trabalho, proporcionando um futuro promissor para aqueles que buscam se aprofundar nessa linguagem.

EducaCiência FastCode para comunidade