**Curso**: ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

**Disciplina**: Algoritmos e Estrutura de Dados

**Identificação da tarefa**: Tarefa 2

**Pontuação**: 10 pontos de 100

**TAREFA 2**

Envio de Arquivo

**Proposta de Atividade**

Bem-vindo(a) à atividade da Unidade 2! Preparado(a) para elevar suas habilidades de programação ao próximo nível? Esta série de exercícios foi planejada para fortalecer seu entendimento dos conceitos essenciais de programação de maneira prática e envolvente.

Aqui, você vai explorar e aplicar uma variedade de temas abordados ao longo da unidade, como Listas (ArrayList), Pilhas, Filas, Maps (HashMap) e Algoritmos de Ordenação. Cada exercício traz um desafio único para te ajudar a dominar esses conceitos fundamentais.

Vamos juntos nessa jornada de aprendizado? Bons estudos e mãos à obra!

```java

Compilar o código:

javac src/Exercicio\_1.java //(isso criará o arquivo Exercicio\_1.class dentro da pasta src).

Executar o programa:

java -cp src Exercicio\_1 //(o -cp src informa ao Java para procurar o arquivo .class dentro do diretório src).

```

Listas (ArrayList):

## Exercicio 1 - Inversão de Lista:

Imagine que você está desenvolvendo um recurso em um aplicativo que exibe uma lista de tarefas do dia em ordem inversa. Escreva um programa que inverta os elementos de uma lista de tarefas (ArrayList) sem utilizar métodos prontos da linguagem, dando ao usuário a opção de ver a lista do fim para o começo.

```java

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Solução para o Exercício 1 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Escreva um programa que inverta os elementos de uma lista de

\* tarefas (ArrayList) sem utilizar métodos prontos da linguagem, dando ao

\* usuário a opção de ver a lista do fim para o começo.

\*

\* "Além da solução padrão solicitada, criei também uma versão `Exercicio\_1\_Dialogo.java` com interface

\* gráfica (JOptionPane) para demonstrar a aplicação do algoritmo em um contexto com interação visual

\* com o usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_1 {

public static void main(String[] args) {

// Força a quebra de linha inicial no terminal

System.out.println();

// 1. Criamos uma lista de tarefas original.

List<String> tarefas = new ArrayList<>();

tarefas.add("Estudar Algoritmos");

tarefas.add("Fazer o exercício 1");

tarefas.add("Preparar o jantar");

tarefas.add("Ler um livro");

tarefas.add("Assistir a uma aula");

// 2. Exibimos a lista original com a nova formatação.

System.out.println("===========================================");

System.out.println(" App de Tarefas - Lista Original");

System.out.println("===========================================");

for (String tarefa : tarefas) {

System.out.println("- " + tarefa);

}

System.out.println("===========================================");

System.out.println(); // Adiciona uma linha em branco para espaçamento

// 3. Chamamos o método para inverter a lista.

List<String> tarefasInvertidas = inverterLista(tarefas);

// 4. Exibimos a nova lista, agora invertida e formatada.

System.out.println("===========================================");

System.out.println(" App de Tarefas - Lista Invertida");

System.out.println("===========================================");

for (String tarefa : tarefasInvertidas) {

System.out.println("- " + tarefa);

}

System.out.println("===========================================");

System.out.println(); // Adiciona uma linha em branco para espaçamento

}

/\*\*

\* Inverte uma lista de Strings criando uma nova lista.

\*

\* @param listaOriginal A lista a ser invertida.

\* @return Uma nova lista contendo os elementos da lista original em ordem

\* inversa.

\*/

public static List<String> inverterLista(List<String> listaOriginal) {

// Criamos uma nova lista vazia que armazenará o resultado.

List<String> listaInvertida = new ArrayList<>();

// Iteramos sobre a lista original, começando do último elemento (índice

// tamanho - 1)

// até o primeiro elemento (índice 0).

for (int i = listaOriginal.size() - 1; i >= 0; i--) {

// Para cada elemento, o adicionamos à nova lista.

// Como estamos percorrendo a original de trás para frente,

// os elementos são inseridos na nova lista em ordem inversa.

listaInvertida.add(listaOriginal.get(i));

}

// Retornamos a nova lista que contém a ordem invertida.

return listaInvertida;

}

}

```

## Exercicio 2 - Remoção de Elementos Duplicados:

Em um sistema de cadastro, é comum que contatos sejam adicionados mais de uma vez. Crie um método que remova contatos duplicados de uma lista de usuários (ArrayList), deixando apenas uma entrada por pessoa e otimizando o banco de dados.

```java

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Solução para o Exercício 2 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Crie um método que remova contatos duplicados de uma lista de

\* usuários (ArrayList), deixando apenas uma entrada por pessoa e otimizando o

\* banco de dados.

\*

\* "Além da solução padrão solicitada, criei também uma versão `Exercicio\_2\_Dialogo.java`com interface \* gráfica (JOptionPane) para demonstrar a aplicação do algoritmo em um contexto com interação visual \* com o usuário, explorando \* outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_2 {

public static void main(String[] args) {

// Força a quebra de linha inicial no terminal

System.out.println();

// 1. Criamos uma lista de contatos com várias duplicatas.

List<String> contatos = new ArrayList<>();

contatos.add("Ana");

contatos.add("Bruno");

contatos.add("Ana");

contatos.add("Carlos");

contatos.add("Daniela");

contatos.add("Bruno");

contatos.add("Ana");

// 2. Exibimos a lista original com a formatação padrão.

System.out.println("===========================================");

System.out.println(" Sistema de Contatos - Lista Original");

System.out.println("===========================================");

for (String contato : contatos) {

System.out.println("- " + contato);

}

System.out.println("===========================================");

System.out.println(); // Adiciona uma linha em branco para espaçamento

// 3. Chamamos o método para remover os elementos duplicados.

List<String> contatosSemDuplicatas = removerDuplicatas(contatos);

// 4. Exibimos a nova lista, sem as duplicatas.

System.out.println("===========================================");

System.out.println(" Sistema de Contatos - Lista Corrigida");

System.out.println("===========================================");

for (String contato : contatosSemDuplicatas) {

System.out.println("- " + contato);

}

System.out.println("===========================================");

System.out.println(); // Adiciona uma linha em branco para espaçamento

}

/\*\*

\* Remove elementos duplicados de uma lista de Strings usando iteração manual.

\*

\* @param listaOriginal A lista contendo possíveis duplicatas.

\* @return Uma nova lista contendo apenas elementos únicos, na ordem em que

\* apareceram pela primeira vez.

\*/

public static List<String> removerDuplicatas(List<String> listaOriginal) {

// Criamos uma nova lista vazia que armazenará o resultado.

List<String> listaSemDuplicatas = new ArrayList<>();

// Usamos um "for-each loop" para percorrer cada elemento da lista original.

for (String elemento : listaOriginal) {

// Verificamos se a nova lista (listaSemDuplicatas) JÁ CONTÉM o elemento

// atual.

if (!listaSemDuplicatas.contains(elemento)) {

// Se o elemento AINDA NÃO EXISTE na nova lista, nós o adicionamos.

listaSemDuplicatas.add(elemento);

}

// Se o elemento já existe, simplesmente o ignoramos e continuamos para o

// próximo.

}

// Retornamos a nova lista, que agora contém apenas os elementos únicos.

return listaSemDuplicatas;

}

}

```

## Exercicio 3 - Mesclar Listas:

Suponha que você está gerenciando duas listas de convidados para um evento e deseja criar uma nova lista com os nomes intercalados das listas originais. Implemente uma função que receba duas listas de convidados e retorne uma nova lista com os elementos intercalados.

Pilhas:

```java

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Solução para o Exercício 3 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Implemente uma função que receba duas listas de convidados e

\* retorne uma nova lista com os elementos intercalados.

\*

\* "Além da solução padrão solicitada, criei também uma versão `Exercicio\_3\_Dialogo.java` com interface

\* gráfica (JOptionPane) para demonstrar a aplicação do algoritmo em um contexto com interação visual

\* com o usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_3 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial

// 1. Criamos as duas listas de convidados.

// A Lista 2 é intencionalmente maior para testar a lógica.

List<String> convidadosLista1 = new ArrayList<>();

convidadosLista1.add("Carlos");

convidadosLista1.add("Sandra");

convidadosLista1.add("Roberto");

List<String> convidadosLista2 = new ArrayList<>();

convidadosLista2.add("Mariana");

convidadosLista2.add("Felipe");

convidadosLista2.add("Juliana");

convidadosLista2.add("Ricardo");

convidadosLista2.add("Patrícia");

// 2. Exibimos as listas originais.

System.out.println("===========================================");

System.out.println(" Lista de Convidados 1");

System.out.println("===========================================");

for (String convidado : convidadosLista1) {

System.out.println("- " + convidado);

}

System.out.println("===========================================");

System.out.println();

System.out.println("===========================================");

System.out.println(" Lista de Convidados 2");

System.out.println("===========================================");

for (String convidado : convidadosLista2) {

System.out.println("- " + convidado);

}

System.out.println("===========================================");

System.out.println();

// 3. Chamamos o método para intercalar as listas.

List<String> listaFinal = intercalarListas(convidadosLista1, convidadosLista2);

// 4. Exibimos a lista final intercalada.

System.out.println("===========================================");

System.out.println(" Lista Final de Convidados");

System.out.println(" (Listas Intercaladas)");

System.out.println("===========================================");

for (String convidado : listaFinal) {

System.out.println("- " + convidado);

}

System.out.println("===========================================");

System.out.println();

}

/\*\*

\* Intercala os elementos de duas listas em uma nova lista.

\*

\* A intercalação ocorre adicionando um elemento da lista1, seguido por um da

\* lista2, até que todos os elementos de ambas as listas tenham sido adicionados.

\*

\* @param lista1 A primeira lista de convidados.

\* @param lista2 A segunda lista de convidados.

\* @return Uma nova lista com os elementos de lista1 e lista2 intercalados.

\*/

public static List<String> intercalarListas(List<String> lista1, List<String> lista2) {

List<String> listaIntercalada = new ArrayList<>();

int tamanhoMaximo = Math.max(lista1.size(), lista2.size());

for (int i = 0; i < tamanhoMaximo; i++) {

// Adiciona o elemento da primeira lista, se ainda houver.

if (i < lista1.size()) {

listaIntercalada.add(lista1.get(i));

}

// Adiciona o elemento da segunda lista, se ainda houver.

if (i < lista2.size()) {

listaIntercalada.add(lista2.get(i));

}

}

return listaIntercalada;

}

}

```

## Exercicio 4 - Verificação de Expressões:

No desenvolvimento de uma calculadora, você precisa verificar se a expressão matemática digitada está com os parênteses corretamente balanceados. Utilize uma pilha para desenvolver um programa que verifique essa estrutura, ajudando a identificar possíveis erros antes do cálculo.

```java

import java.util.ArrayDeque;

import java.util.Deque;

/\*\*

\* Solução para o Exercício 4 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Utilize uma pilha para desenvolver um programa que verifique se os

\* parênteses em uma expressão matemática estão corretamente balanceados.

\*

\* "Além da solução padrão solicitada, criei também uma versão `Exercicio\_4\_Dialogo.java` com interface

\* gráfica (JOptionPane) para demonstrar a aplicação do algoritmo em um contexto com interação visual

\* com o usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_4 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial

// Lista de expressões para teste

String[] expressoes = {

"((a+b) \* (c-d))", // Válida

"(a+b) \* c)", // Inválida - fecha parêntese a mais

"((a+b) \* c", // Inválida - falta fechar parêntese

"())a+b(()", // Inválida - ordem errada e falta fechar

"a + b", // Válida - sem parênteses

"(()())" // Válida

};

System.out.println("===========================================");

System.out.println(" Verificador de Parênteses Balanceados");

System.out.println("===========================================");

// Itera sobre as expressões e verifica cada uma

for (String exp : expressoes) {

boolean balanceada = verificarParenteses(exp);

System.out.printf("Expressão \"%s\" -> Balanceada? %s\n", exp, balanceada);

}

System.out.println("===========================================");

System.out.println();

}

/\*\*

\* Verifica se os parênteses em uma expressão estão balanceados usando uma Pilha (Deque).

\*

\* @param expressao A string da expressão a ser verificada.

\* @return true se os parênteses estiverem balanceados, false caso contrário.

\*/

public static boolean verificarParenteses(String expressao) {

// Usamos Deque como a implementação da nossa pilha, conforme boas práticas.

Deque<Character> pilha = new ArrayDeque<>();

// Percorremos cada caractere da expressão.

for (char caractere : expressao.toCharArray()) {

// Se for um parêntese de abertura, empilhamos.

if (caractere == '(') {

pilha.push(caractere);

}

// Se for um parêntese de fechamento...

else if (caractere == ')') {

// ...verificamos se a pilha está vazia.

// Se estiver, significa que há um ')' a mais, então a expressão é inválida.

if (pilha.isEmpty()) {

return false;

}

// Se a pilha não estiver vazia, desempilhamos um '('.

pilha.pop();

}

}

// Ao final, a expressão está balanceada se a pilha estiver vazia.

// Se sobrar algo na pilha, significa que há um '(' a mais.

return pilha.isEmpty();

}

}

```

## Exercicio 5 - Inversão com Pilha:

Suponha que você está desenvolvendo uma ferramenta que permite ao usuário visualizar uma lista de produtos na ordem inversa da inserção original. Implemente um método que utilize uma pilha para inverter a ordem dos elementos de uma lista de produtos (ArrayList).

Filas:

```java

import java.util.ArrayList;

import java.util.Stack;

/\*\*

\* Solução para o Exercício 5 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Crie um programa que inverta a ordem dos elementos em um ArrayList

\* de inteiros utilizando uma pilha como estrutura auxiliar.

\*

\* "Além da solução padrão solicitada, criei também uma versão `Exercicio\_5\_Dialogo.java` com interface

\* gráfica (JOptionPane) para demonstrar a aplicação do algoritmo em um contexto com interação visual

\* com o usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_5 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial

// Cria e preenche a lista de números.

ArrayList<Integer> numeros = new ArrayList<>();

numeros.add(1);

numeros.add(2);

numeros.add(3);

numeros.add(4);

numeros.add(5);

System.out.println("===========================================");

System.out.println(" Inversor de ArrayList com Pilha");

System.out.println("===========================================");

System.out.println("Lista Original: " + numeros);

System.out.println();

// Inverte a lista usando o método auxiliar

inverterArrayListComPilha(numeros);

System.out.println("Lista Invertida: " + numeros);

System.out.println("===========================================");

System.out.println();

}

/\*\*

\* Inverte a ordem dos elementos de um ArrayList de inteiros usando uma Pilha.

\*

\* @param lista O ArrayList de inteiros a ser invertido.

\*/

public static void inverterArrayListComPilha(ArrayList<Integer> lista) {

// 1. Cria uma pilha para servir como estrutura de dados auxiliar.

Stack<Integer> pilha = new Stack<>();

// 2. Empurra todos os elementos da lista para a pilha.

// O primeiro elemento da lista (índice 0) ficará no fundo da pilha,

// e o último elemento ficará no topo.

for (Integer numero : lista) {

pilha.push(numero);

}

// 3. Substitui os elementos na lista original pelos elementos da pilha.

// O método pop() remove e retorna o elemento do topo da pilha (LIFO).

// Assim, o último elemento a entrar será o primeiro a sair, invertendo a ordem.

for (int i = 0; i < lista.size(); i++) {

lista.set(i, pilha.pop());

}

}

}

```

## Exercicio 6 - Implementação de um Sistema de Fila de Espera:

Em uma clínica, os pacientes aguardam atendimento em uma fila de espera. Crie um sistema que simule essa fila, permitindo adicionar, remover e exibir a ordem de atendimento dos pacientes. Isso ajudará na organização e no controle da ordem de atendimento.

```java

import java.util.InputMismatchException;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

/\*\*

\* Solução para o Exercício 6 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Crie um sistema que simule uma fila de espera de pacientes em uma

\* clínica, permitindo adicionar, remover e exibir a ordem de atendimento.

\*

\* "Além da solução padrão solicitada, criarei também uma versão

\* `Exercicio\_6\_Dialogo.java` com interface gráfica (JOptionPane) para

\* demonstrar a aplicação do algoritmo em um contexto com interação visual com o

\* usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_6 {

// Scanner e Fila declarados como estáticos para serem acessíveis em toda a classe.

private static final Scanner scanner = new Scanner(System.in);

private static final Queue<String> filaDePacientes = new LinkedList<>();

public static void main(String[] args) {

boolean executando = true;

// Loop principal do menu. Continua executando até que o usuário escolha sair.

while (executando) {

exibirMenu();

int opcao = lerOpcao();

switch (opcao) {

case 1:

adicionarPaciente();

break;

case 2:

chamarProximo();

break;

case 3:

exibirFila();

break;

case 4:

executando = false;

break;

default:

System.out.println("\nOpção inválida. Por favor, escolha um número entre 1 e 4.");

}

}

System.out.println("\n=================================================");

System.out.println(" Sistema de Fila da Clínica Finalizado");

System.out.println("=================================================");

System.out.println();

scanner.close(); // Fecha o scanner para liberar recursos.

}

/\*\*

\* Exibe o menu de opções formatado no console.

\*/

private static void exibirMenu() {

System.out.println("\n=================================================");

System.out.println(" Sistema de Fila de Espera da Clínica");

System.out.println("=================================================");

System.out.println("1. Adicionar novo paciente à fila");

System.out.println("2. Chamar próximo paciente para atendimento");

System.out.println("3. Exibir fila de pacientes atual");

System.out.println("4. Sair do sistema");

System.out.println("=================================================");

System.out.print("Escolha uma opção: ");

}

/\*\*

\* Lê e valida a entrada numérica do usuário para a opção do menu.

\*

\* @return O número da opção escolhida, ou -1 se a entrada for inválida.

\*/

private static int lerOpcao() {

try {

int opcao = scanner.nextInt();

scanner.nextLine(); // Consome a quebra de linha pendente.

return opcao;

} catch (InputMismatchException e) {

System.out.println("\nErro: Entrada inválida. Por favor, digite um número.");

scanner.nextLine(); // Limpa o buffer do scanner para a próxima leitura.

return -1; // Retorna um valor inválido para repetir o loop.

}

}

/\*\*

\* Solicita o nome de um paciente e o adiciona ao final da fila.

\*/

private static void adicionarPaciente() {

System.out.print("\nDigite o nome completo do paciente: ");

String nome = scanner.nextLine();

filaDePacientes.add(nome); // Adiciona o paciente ao FINAL da fila.

System.out.println("Paciente \"" + nome + "\" foi adicionado à fila com sucesso.");

}

/\*\*

\* Remove e exibe o próximo paciente do início da fila.

\*/

private static void chamarProximo() {

if (filaDePacientes.isEmpty()) {

System.out.println("\nA fila de espera está vazia. Não há pacientes para chamar.");

} else {

// O método poll() remove e retorna o elemento no INÍCIO da fila.

String proximo = filaDePacientes.poll();

System.out.println("\nPróximo paciente chamado para atendimento: \"" + proximo + "\"");

}

}

/\*\*

\* Exibe todos os pacientes na fila na ordem de chegada, sem removê-los.

\*/

private static void exibirFila() {

System.out.println("\n----------- FILA DE ESPERA ATUAL -----------");

if (filaDePacientes.isEmpty()) {

System.out.println("A fila está vazia.");

} else {

int posicao = 1;

// Itera sobre a fila para mostrar cada paciente e sua posição.

for (String paciente : filaDePacientes) {

System.out.println(posicao + ". " + paciente);

posicao++;

}

}

System.out.println("--------------------------------------------");

}

}

```

## Exercicio 7 - Ordem de Chegada:

Imagine que você está organizando o fluxo de clientes em uma loja durante uma promoção. Crie um programa que simule a ordem de chegada e exiba a ordem de saída, respeitando a fila de atendimento.

Maps (HashMap):

```java

import java.util.LinkedList;

import java.util.Queue;

import java.util.List;

import java.util.Arrays;

/\*\*

\* Solução para o Exercício 7 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Crie um programa que simule a ordem de chegada e exiba a ordem

\* de saída de clientes em uma loja, respeitando a fila de atendimento.

\*

\* "Além da solução padrão solicitada, criarei também uma versão

\* `Exercicio\_7\_Dialogo.java` com interface gráfica (JOptionPane) para

\* demonstrar a aplicação do algoritmo em um contexto com interação visual com o

\* usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério

\*/

public class Exercicio\_7 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial no terminal

// A fila de clientes, implementada com LinkedList.

Queue<String> filaClientes = new LinkedList<>();

// Lista de clientes que chegarão à loja para a simulação.

List<String> clientesChegando = Arrays.asList("Mariana", "Felipe", "Juliana", "Ricardo", "Patrícia");

// --- 1. Simulação da Chegada dos Clientes ---

System.out.println("===============================================");

System.out.println(" LOJA ABERTA - Início da Simulação de Fila");

System.out.println("===============================================");

System.out.println();

System.out.println("--- Chegada dos Clientes ---");

for (String cliente : clientesChegando) {

filaClientes.add(cliente); // Adiciona o cliente ao final da fila.

System.out.println("-> O cliente '" + cliente + "' chegou e entrou na fila.");

}

System.out.println();

System.out.println("Fila de espera atual: " + filaClientes);

System.out.println();

// --- 2. Simulação do Atendimento (Saída dos Clientes) ---

System.out.println("--- Atendimento dos Clientes (Ordem de Saída) ---");

int clienteNumero = 1;

// O loop continua enquanto houver clientes na fila.

while (!filaClientes.isEmpty()) {

// poll() remove e retorna o cliente do início da fila (o que chegou primeiro).

String clienteAtendido = filaClientes.poll();

System.out.println("<- Atendimento #" + clienteNumero + ": O cliente '" + clienteAtendido + "' foi atendido e saiu.");

clienteNumero++;

}

System.out.println();

System.out.println("Todos os clientes foram atendidos!");

System.out.println("Fila de espera atual: " + filaClientes);

System.out.println();

System.out.println("===============================================");

System.out.println(" FIM DA SIMULAÇÃO - Loja Fechada");

System.out.println("===============================================");

System.out.println();

}

}

```

## Exercicio 8 - Contagem de Palavras:

Você precisa criar uma análise de palavras usadas em um artigo para entender quais termos aparecem com maior frequência. Escreva um programa que conte a frequência de cada palavra em uma frase, utilizando um HashMap, para identificar as palavras mais comuns.

```java

import java.util.HashMap;

import java.util.Map;

/\*\*

\* Solução para o Exercício 8 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Escreva um programa que conte a frequência de cada palavra em uma

\* frase, utilizando um HashMap, para identificar as palavras mais comuns.

\*

\* "Além da solução padrão solicitada, criarei também uma versão

\* `Exercicio\_8\_Dialogo.java` com interface gráfica (JOptionPane) para

\* demonstrar a aplicação do algoritmo em um contexto com interação visual com o

\* usuário, explorando outros recursos da plataforma Java."

\*

\* Autor: Rogério (com assistência do Gemini)

\*/

public class Exercicio\_8 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial no terminal

// 1. Artigo de exemplo para análise.

String artigo = "A persistência é o caminho do êxito. O estudo constante e a prática dedicada são o segredo do sucesso. Estudo e prática levam à perfeição.";

// 2. Exibimos o artigo original.

System.out.println("========================================================");

System.out.println(" Analisador de Frequência de Palavras");

System.out.println("========================================================");

System.out.println();

System.out.println("--- Artigo Original ---");

System.out.println(artigo);

System.out.println();

// 3. Chamamos o método para contar a frequência das palavras.

Map<String, Integer> frequenciaPalavras = contarFrequencia(artigo);

// 4. Exibimos o resultado da contagem.

System.out.println("--- Frequência de Cada Palavra ---");

// A interface Map.forEach simplifica a iteração sobre as entradas do mapa.

frequenciaPalavras.forEach((palavra, contagem) -> {

System.out.println("'" + palavra + "': " + contagem + " vez(es)");

});

System.out.println();

System.out.println("========================================================");

System.out.println(" Análise Concluída");

System.out.println("========================================================");

System.out.println();

}

/\*\*

\* Conta a frequência de cada palavra em um texto.

\*

\* @param texto O texto a ser analisado.

\* @return Um Map onde a chave é a palavra e o valor é sua frequência.

\*/

public static Map<String, Integer> contarFrequencia(String texto) {

// O HashMap armazenará a palavra (chave) e sua contagem (valor).

Map<String, Integer> mapaFrequencia = new HashMap<>();

// 1. Normalização do texto:

// - Converte tudo para minúsculas para não diferenciar "A" de "a".

// - Remove pontuações como pontos e vírgulas para simplificar a análise.

String textoNormalizado = texto.toLowerCase().replaceAll("[.,]", "");

// 2. Divisão do texto em palavras:

// - O método split("\s+") divide a string por um ou mais espaços em branco.

String[] palavras = textoNormalizado.split("\s+");

// 3. Contagem das palavras:

// - Itera sobre cada palavra no array de palavras.

for (String palavra : palavras) {

// Ignora strings vazias que podem surgir de múltiplos espaços.

if (!palavra.isEmpty()) {

// Usa o método getOrDefault para simplificar a lógica:

// - Se a palavra já existe no mapa, pega o valor atual.

// - Se não existe, usa o valor padrão (0).

// Em seguida, adiciona 1 ao valor e atualiza o mapa.

mapaFrequencia.put(palavra, mapaFrequencia.getOrDefault(palavra, 0) + 1);

}

}

return mapaFrequencia;

}

}

```

## Exercicio 9 - Mesclar Mapas:

Em um sistema de inventário, você tem dois registros de produtos e deseja mesclar as informações. Implemente um método para mesclar dois mapas (HashMaps) de produtos, cuidando para tratar possíveis conflitos de chaves.

Algoritmos de Ordenação:

```java

import java.util.HashMap;

import java.util.Map;

/\*\*

\* Solução para o Exercício 9 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Implemente um método para mesclar dois mapas (HashMaps) de

\* produtos, cuidando para tratar possíveis conflitos de chaves (somando os

\* valores).

\*

\* "Além da solução padrão solicitada, criarei também uma versão

\* `Exercicio\_9\_Dialogo.java` com interface gráfica (JOptionPane) para

\* demonstrar a aplicação do algoritmo em um contexto com interação visual com o

\* usuário."

\*

\* Autor: Rogério

\*/

public class Exercicio\_9 {

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial no terminal

// 1. Criação dos inventários das duas lojas.

Map<String, Integer> inventarioLojaA = new HashMap<>();

inventarioLojaA.put("Maçã", 50);

inventarioLojaA.put("Banana", 30);

inventarioLojaA.put("Laranja", 25);

Map<String, Integer> inventarioLojaB = new HashMap<>();

inventarioLojaB.put("Banana", 40); // Conflito de chave

inventarioLojaB.put("Pera", 20);

inventarioLojaB.put("Maçã", 15); // Conflito de chave

// 2. Exibição dos inventários originais.

System.out.println("========================================================");

System.out.println(" Sistema de Consolidação de Inventário");

System.out.println("========================================================");

System.out.println();

exibirInventario("Inventário da Loja A", inventarioLojaA);

exibirInventario("Inventário da Loja B", inventarioLojaB);

// 3. Mescla os dois inventários.

Map<String, Integer> inventarioConsolidado = mesclarInventarios(inventarioLojaA, inventarioLojaB);

// 4. Exibição do inventário consolidado.

exibirInventario("Inventário Consolidado (Loja A + Loja B)", inventarioConsolidado);

System.out.println("========================================================");

System.out.println(" Consolidação Concluída");

System.out.println("========================================================");

System.out.println();

}

/\*\*

\* Mescla dois mapas de inventário, somando as quantidades em caso de chaves duplicadas.

\*

\* @param inventarioA O primeiro inventário.

\* @param inventarioB O segundo inventário.

\* @return Um novo mapa contendo o inventário consolidado.

\*/

public static Map<String, Integer> mesclarInventarios(Map<String, Integer> inventarioA, Map<String, Integer> inventarioB) {

// Cria um novo mapa já com todos os itens do primeiro inventário.

Map<String, Integer> inventarioConsolidado = new HashMap<>(inventarioA);

// Itera sobre o segundo inventário.

inventarioB.forEach((produto, quantidade) -> {

// O método merge() simplifica a lógica de conflito:

// - Se a chave (produto) não existe no mapa consolidado, ela é inserida com sua quantidade.

// - Se a chave já existe, a função (Integer::sum) é aplicada aos valores (antigo e novo).

inventarioConsolidado.merge(produto, quantidade, Integer::sum);

});

return inventarioConsolidado;

}

/\*\*

\* Método auxiliar para exibir o conteúdo de um inventário de forma formatada.

\*

\* @param titulo O título a ser exibido para o inventário.

\* @param inventario O mapa do inventário a ser exibido.

\*/

private static void exibirInventario(String titulo, Map<String, Integer> inventario) {

System.out.println("--------------------------------------------------------");

System.out.println(" " + titulo);

System.out.println("--------------------------------------------------------");

if (inventario.isEmpty()) {

System.out.println("Inventário vazio.");

} else {

inventario.forEach((produto, quantidade) -> {

System.out.printf("- %-10s: %d unidades\n", produto, quantidade);

});

}

System.out.println("--------------------------------------------------------");

System.out.println();

}

}

```

## Exercicio 10 - Implementação de Algoritmo de Ordenação:

Imagine que você está desenvolvendo um sistema de classificação que exibe os produtos mais populares em uma loja online. Escolha um algoritmo de ordenação (ex: Bubble Sort, Quick Sort, Merge Sort) e implemente-o para ordenar uma lista de produtos com base nas vendas, de forma decrescente, para que os mais vendidos apareçam primeiro.

```java

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Solução para o Exercício 10 da Unidade 02 de Algoritmos e Estrutura de Dados.

\*

\* Descrição: Implementa um algoritmo de ordenação (Merge Sort) para classificar

\* uma lista de produtos com base no número de vendas, em ordem decrescente.

\*

\* "Além da solução padrão solicitada, criarei também uma versão

\* `Exercicio\_10\_Dialogo.java` com interface gráfica (JOptionPane) para

\* demonstrar a aplicação do algoritmo."

\*

\* Autor: Rogério

\*/

public class Exercicio\_10 {

/\*\*

\* Classe interna para representar um Produto com nome e número de vendas.

\*/

static class Produto {

String nome;

int vendas;

public Produto(String nome, int vendas) {

this.nome = nome;

this.vendas = vendas;

}

@Override

public String toString() {

return String.format("%20s | Vendas: %d", nome, vendas);

}

}

public static void main(String[] args) {

System.out.println(); // Espaçamento inicial

// 1. Cria uma lista de produtos desordenada.

List<Produto> produtos = new ArrayList<>();

produtos.add(new Produto("Smart TV 50\"", 120));

produtos.add(new Produto("Notebook Gamer", 85));

produtos.add(new Produto("Smartphone Pro", 350));

produtos.add(new Produto("Cadeira de Escritório", 210));

produtos.add(new Produto("Fone Bluetooth", 400));

produtos.add(new Produto("Teclado Mecânico", 150));

System.out.println("========================================================");

System.out.println(" Sistema de Classificação de Produtos Populares");

System.out.println("========================================================");

System.out.println();

// 2. Exibe a lista original.

exibirLista("Lista de Produtos Original (Desordenada)", produtos);

// 3. Ordena a lista usando Merge Sort.

List<Produto> produtosOrdenados = mergeSort(produtos);

// 4. Exibe a lista ordenada.

exibirLista("Produtos Mais Populares (Ordenados por Vendas)", produtosOrdenados);

System.out.println("========================================================");

System.out.println(" Classificação Concluída");

System.out.println("========================================================");

System.out.println();

}

/\*\*

\* Ordena uma lista de Produtos usando o algoritmo Merge Sort (recursivo).

\*

\* @param lista A lista a ser ordenada.

\* @return Uma nova lista ordenada em ordem decrescente de vendas.

\*/

public static List<Produto> mergeSort(List<Produto> lista) {

// Caso base: se a lista tem 1 ou 0 elementos, já está ordenada.

if (lista.size() <= 1) {

return lista;

}

// Divide a lista em duas metades.

int meio = lista.size() / 2;

List<Produto> metadeEsquerda = new ArrayList<>(lista.subList(0, meio));

List<Produto> metadeDireita = new ArrayList<>(lista.subList(meio, lista.size()));

// Chama recursivamente o mergeSort para cada metade.

metadeEsquerda = mergeSort(metadeEsquerda);

metadeDireita = mergeSort(metadeDireita);

// Combina (merge) as duas metades ordenadas.

return merge(metadeEsquerda, metadeDireita);

}

/\*\*

\* Combina duas listas já ordenadas em uma única lista ordenada.

\*

\* @param esquerda Lista da esquerda, já ordenada.

\* @param direita Lista da direita, já ordenada.

\* @return A lista combinada e totalmente ordenada.

\*/

private static List<Produto> merge(List<Produto> esquerda, List<Produto> direita) {

List<Produto> resultado = new ArrayList<>();

int i = 0, j = 0; // Ponteiros para as listas da esquerda e direita

// Compara os elementos das duas listas e adiciona o maior ao resultado.

while (i < esquerda.size() && j < direita.size()) {

// Para ordem DECRESCENTE, comparamos se o da esquerda é MAIOR que o da direita.

if (esquerda.get(i).vendas >= direita.get(j).vendas) {

resultado.add(esquerda.get(i));

i++;

} else {

resultado.add(direita.get(j));

j++;

}

}

// Adiciona os elementos restantes de qualquer uma das listas (se houver).

while (i < esquerda.size()) {

resultado.add(esquerda.get(i));

i++;

}

while (j < direita.size()) {

resultado.add(direita.get(j));

j++;

}

return resultado;

}

/\*\*

\* Método auxiliar para exibir a lista de produtos de forma formatada.

\*/

private static void exibirLista(String titulo, List<Produto> produtos) {

System.out.println("--------------------------------------------------------");

System.out.println(" " + titulo);

System.out.println("--------------------------------------------------------");

for (Produto produto : produtos) {

System.out.println(produto);

}

System.out.println("--------------------------------------------------------");

System.out.println();

}

}

```

**Instruções de Entrega**

* Realize todos os exercícios da lista na linguagem Java e aplique os conceitos solicitados em cada questão.
* Envie o código para análise, pode ser por repositório do GitHub ou a pasta compactada com seu código.