**Tugas Praktikum Kelompok ke-2**

**Struktur Data dan Analisis Alogima**

**Minggu 7**



YAYANG DIKI HERMAWAN (2802517840)

ANDRIAN RAIHANNUDIN (2802516390)

ELIZABETH IMMANUELA PUTRI (2802552130)

SOFIA IFTITA NUR RAHMANINGRUM (2802549760)

HELLA DWI PRATIWI (2802519650)

**LAPORAN STRUKTUR DATA: QUEUE, STACK, DAN LINKED LIST**

**1. Pendahuluan**

Struktur data merupakan komponen penting dalam pemrograman karena berperan dalam penyimpanan dan pengelolaan data. Tiga struktur data yang umum digunakan adalah Queue, Stack, dan Linked List. Laporan ini membahas metode, implementasi kode, efisiensi waktu eksekusi, serta analisis kelebihan dan kekurangan masing-masing struktur data.

**2. Metode per Struktur Data**

2.1 Queue

Queue (antrian) adalah struktur data linear yang menerapkan prinsip FIFO (First-In-First-Out). Elemen pertama yang dimasukkan akan menjadi elemen pertama yang keluar.

2.2 Stack

Stack (tumpukan) menggunakan prinsip LIFO (Last-In-First-Out). Elemen terakhir yang dimasukkan adalah yang pertama kali dikeluarkan.

2.3 Linked List

Linked List adalah struktur data yang terdiri dari simpul (node) yang saling terhubung. Setiap simpul memiliki data dan referensi ke simpul berikutnya. Tidak seperti array, Linked List tidak memerlukan alokasi memori tetap.

**3. Implementasi Kode (Java)**

3.1 Queue

package Queue;

import java.util.Scanner;

// Class Node untuk merepresentasikan setiap pelanggan dalam antrian

class Node {

String nama;

Node next;

Node(String nama) {

this.nama = nama;

this.next = null;

}

}

// Class Queue untuk mengelola antrian pelanggan

class Queue {

Node front, rear;

Queue() {

front = rear = null; // Antrian awalnya kosong

}

// Menambah pelanggan ke antrian

void enqueue(String nama) {

Node baru = new Node(nama);

if (rear == null) {

// Kalau antrian kosong, front dan rear menunjuk ke pelanggan baru

front = rear = baru;

} else {

// Tambahkan pelanggan baru di belakang antrian

rear.next = baru;

rear = baru;

}

System.out.println("[System] " + nama + " berhasil bergabung dalam antrean. Silakan menunggu!");

}

// Melayani pelanggan (menghapus dari depan antrian)

void dequeue() {

if (front == null) {

System.out.println("[System] Antrian kosong. Tidak ada pelanggan yang bisa dilayani.");

return;

}

System.out.println("[System] Saat ini sedang melayani pelanggan: " + front.nama);

front = front.next; // Pindahkan front ke pelanggan berikutnya

if (front == null) {

rear = null; // Kalau habis, set rear jadi null juga

}

// Menampilkan daftar antrian setelah melayani

display();

}

// Menampilkan semua pelanggan dalam antrian

void display() {

if (front == null) {

System.out.println("[System] Saat ini tidak ada pelanggan dalam antrean.");

return;

}

System.out.println("\n===== Pelanggan dalam antrean =====");

Node temp = front;

int nomor = 1;

while (temp != null) {

System.out.println(nomor + ". " + temp.nama);

temp = temp.next;

nomor++;

}

}

}

public class CustomerService {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

Queue customerQueue = new Queue(); // Membuat antrian baru

int pilihan;

System.out.println("\n===== Selamat Datang di Layanan Customer Service Kantin A =====");

do {

System.out.println("\nMenu:");

System.out.println("1. Daftar Antrian Baru");

System.out.println("2. Layani Pelanggan");

System.out.println("3. Cek Antrian Saat Ini");

System.out.println("4. Keluar");

System.out.print("Pilihan Anda: ");

pilihan = input.nextInt();

input.nextLine(); // Buat ngambil enter setelah nextInt()

switch (pilihan) {

case 1:

System.out.print("Masukkan nama pelanggan: ");

String nama = input.nextLine();

customerQueue.enqueue(nama); // Tambah pelanggan baru

break;

case 2:

customerQueue.dequeue(); // Layani pelanggan

break;

case 3:

customerQueue.display(); // Tampilkan semua pelanggan

break;

case 4:

System.out.println("\n[System] Terima kasih telah menggunakan layanan kami. Sampai jumpa!");

break;

default:

System.out.println("[System] Pilihan tidak valid. Silakan coba lagi.");

}

} while (pilihan != 4);

input.close(); // Tutup Scanner

}

}

3.2 Stack

package Stack;

import java.util.Scanner;

import java.util.Stack;

public class TextEditorSimulator {

public static void main(String[] args) {

Stack<String[]> undoStack = new Stack<>(); // Stack untuk undo

Stack<String[]> redoStack = new Stack<>(); // Stack untuk redo

Scanner input = new Scanner(System.in);

String currentNama = "";

String currentMenu = "";

while (true) {

// Tampilkan pesanan saat ini

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

if (currentNama.isEmpty() && currentMenu.isEmpty()) {

System.out.println("Pesanan anda: -");

} else {

System.out.println("Pesanan anda: " + currentNama + " - " + currentMenu);

}

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

// Menu pilihan

System.out.println("Menu:");

System.out.println("1. Buat Pesanan");

System.out.println("2. Undo");

System.out.println("3. Redo");

System.out.println("4. Batalkan Pesanan");

System.out.println("5. Konfirmasi Pesanan");

System.out.print("Pilih aksi: ");

int choice = input.nextInt();

input.nextLine(); // Buat ambil enter

switch (choice) {

case 1:

// Simpan kondisi sekarang sebelum perubahan

undoStack.push(new String[]{currentNama, currentMenu});

redoStack.clear(); // Reset redo karena ada perubahan baru

System.out.print("Masukkan nama: ");

currentNama = input.nextLine();

System.out.print("Masukkan menu: ");

currentMenu = input.nextLine();

break;

case 2:

// Undo perubahan

if (!undoStack.isEmpty()) {

redoStack.push(new String[]{currentNama, currentMenu});

String[] undoState = undoStack.pop();

currentNama = undoState[0];

currentMenu = undoState[1];

System.out.println("[System] Undo berhasil!");

} else {

System.out.println("[System] Tidak ada yang bisa di-undo!");

}

break;

case 3:

// Redo perubahan

if (!redoStack.isEmpty()) {

undoStack.push(new String[]{currentNama, currentMenu});

String[] redoState = redoStack.pop();

currentNama = redoState[0];

currentMenu = redoState[1];

System.out.println("[System] Redo berhasil!");

} else {

System.out.println("[System] Tidak ada yang bisa di-redo!");

}

break;

case 4:

// Membatalkan pesanan

System.out.println("[System] Pesanan berhasil dibatalkan.");

input.close();

return;

case 5:

System.out.println("\n[System] Pesanan berhasil dibuat!");

if (currentNama.isEmpty() && currentMenu.isEmpty()) {

System.out.println("[System] Tidak ada pesanan yang dipilih.");

} else {

System.out.println("Detail Pesanan:");

System.out.println("Nama : " + currentNama + ", Menu : " + currentMenu);

}

System.out.println("[System] Pesanan berhasil dibuat, silahkan lanjutkan pembayaran. Terima kasih!");

input.close();

return;

default:

System.out.println("[System] Pilihan tidak sesuai, coba lagi ya!");

}

}

}

}

3.3 Linked List

package LinkedList;

import java.util.Scanner;

// Class Mahasiswa untuk menyimpan data mahasiswa

class Mahasiswa {

String nim;

String nama;

int nilai;

Mahasiswa next;

public Mahasiswa(String nim, String nama, int nilai) {

this.nim = nim;

this.nama = nama;

this.nilai = nilai;

this.next = null;

}

}

// Class LinkedListMahasiswa untuk mengelola daftar mahasiswa

class LinkedListMahasiswa {

Mahasiswa head;

public LinkedListMahasiswa() {

this.head = null; // Awal daftar kosong

}

public boolean isEmpty() {

return head == null;

}

// Menambah mahasiswa baru ke akhir list

public void tambahMahasiswa(String nim, String nama, int nilai) {

Mahasiswa mahasiswaBaru = new Mahasiswa(nim, nama, nilai);

if (isEmpty()) {

head = mahasiswaBaru;

} else {

Mahasiswa current = head;

while (current.next != null) {

current = current.next;

}

current.next = mahasiswaBaru;

}

System.out.println("Mahasiswa " + nama + " berhasil ditambahkan.");

}

// Menghapus mahasiswa berdasarkan NIM

public void hapusMahasiswa(String nim) {

if (isEmpty()) {

System.out.println("Daftar mahasiswa kosong.");

return;

}

if (head.nim.equals(nim)) {

String namaTerhapus = head.nama;

head = head.next;

System.out.println("Mahasiswa dengan NIM " + nim + " (" + namaTerhapus + ") berhasil dihapus.");

return;

}

Mahasiswa current = head;

Mahasiswa prev = null;

while (current != null && !current.nim.equals(nim)) {

prev = current;

current = current.next;

}

if (current == null) {

System.out.println("Mahasiswa dengan NIM " + nim + " tidak ditemukan.");

} else {

String namaTerhapus = current.nama;

prev.next = current.next;

System.out.println("Mahasiswa dengan NIM " + nim + " (" + namaTerhapus + ") berhasil dihapus.");

}

}

// Mengupdate nilai mahasiswa

public void updateNilaiMahasiswa(String nim, int nilaiBaru) {

if (isEmpty()) {

System.out.println("Daftar mahasiswa kosong.");

return;

}

Mahasiswa current = head;

while (current != null) {

if (current.nim.equals(nim)) {

current.nilai = nilaiBaru;

System.out.println("Nilai mahasiswa dengan NIM " + nim + " (" + current.nama + ") berhasil diupdate menjadi " + nilaiBaru + ".");

return;

}

current = current.next;

}

System.out.println("Mahasiswa dengan NIM " + nim + " tidak ditemukan.");

}

// Menampilkan daftar seluruh mahasiswa

public void tampilkanDaftarMahasiswa() {

if (isEmpty()) {

System.out.println("Daftar mahasiswa kosong.");

return;

}

System.out.println("\nDaftar Mahasiswa:");

System.out.printf("%-10s %-20s %-5s%n", "NIM", "Nama", "Nilai");

Mahasiswa current = head;

while (current != null) {

System.out.printf("%-10s %-20s %-5d%n", current.nim, current.nama, current.nilai);

current = current.next;

}

System.out.println();

}

}

public class DataMahasiswa {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

LinkedListMahasiswa daftarMahasiswa = new LinkedListMahasiswa();

int maxNIMLength = 10; // Panjang maksimal NIM

while (true) {

System.out.println("Menu:");

System.out.println("1. Tambah Mahasiswa");

System.out.println("2. Hapus Mahasiswa");

System.out.println("3. Update Nilai Mahasiswa");

System.out.println("4. Tampilkan Daftar Mahasiswa");

System.out.println("5. Keluar");

System.out.print("Pilih opsi: ");

String pilihan = scanner.nextLine();

switch (pilihan) {

case "1":

System.out.print("Masukkan NIM: ");

String nimBaru = scanner.nextLine();

if (nimBaru.length() > maxNIMLength || !nimBaru.matches("\\d+")) {

System.out.println("NIM tidak valid. Harus berupa angka dan maksimal " + maxNIMLength + " digit.");

break;

}

System.out.print("Masukkan Nama: ");

String namaBaru = scanner.nextLine();

System.out.print("Masukkan Nilai: ");

int nilaiBaru = scanner.nextInt();

scanner.nextLine(); // Buat ngambil enter

daftarMahasiswa.tambahMahasiswa(nimBaru, namaBaru, nilaiBaru);

break;

case "2":

System.out.print("Masukkan NIM mahasiswa yang ingin dihapus: ");

String nimHapus = scanner.nextLine();

daftarMahasiswa.hapusMahasiswa(nimHapus);

break;

case "3":

System.out.print("Masukkan NIM mahasiswa yang ingin diupdate nilainya: ");

String nimUpdate = scanner.nextLine();

System.out.print("Masukkan nilai baru: ");

int nilaiUpdate = scanner.nextInt();

scanner.nextLine(); // Buat ngambil enter

daftarMahasiswa.updateNilaiMahasiswa(nimUpdate, nilaiUpdate);

break;

case "4":

daftarMahasiswa.tampilkanDaftarMahasiswa();

break;

case "5":

System.out.println("Terima kasih!");

scanner.close();

return;

default:

System.out.println("Opsi tidak valid. Silakan coba lagi.");

}

}

}

}

**4. Analisis Efisiensi Waktu Eksekusi**

package PerformanceTest;

import java.util.Queue;

import java.util.LinkedList;

import java.util.Stack;

public class PerformanceTest {

public static void main(String[] args) {

// Uji performa Queue

Queue<String> queue = new LinkedList<>(); // Menggunakan LinkedList sebagai implementasi Queue

long startQueue = System.nanoTime();

for (int i = 0; i < 1000; i++) {

queue.offer("Pelanggan" + i);

}

long endQueue = System.nanoTime();

long timeQueue = endQueue - startQueue;

// Uji performa Stack

Stack<String[]> undoStack = new Stack<>();

long startStack = System.nanoTime();

for (int i = 0; i < 1000; i++) {

undoStack.push(new String[]{"Nama" + i, "Menu" + i});

}

long endStack = System.nanoTime();

long timeStack = endStack - startStack;

// Uji performa LinkedList Mahasiswa

LinkedList<String> mahasiswaList = new LinkedList<>();

long startLinkedList = System.nanoTime();

for (int i = 0; i < 1000; i++) {

mahasiswaList.add("NIM" + i + " - Nama" + i);

}

long endLinkedList = System.nanoTime();

long timeLinkedList = endLinkedList - startLinkedList;

// Print hasil perbandingan waktu dalam milisecond (ms)

System.out.println("\n===== Hasil Perbandingan Waktu Eksekusi =====");

System.out.println("Queue (Tambah 1000 pelanggan): " + (timeQueue / 1\_000\_000.0) + " ms");

System.out.println("Stack (Tambah 1000 histori): " + (timeStack / 1\_000\_000.0) + " ms");

System.out.println("Linked List (Tambah 1000 mahasiswa): " + (timeLinkedList / 1\_000\_000.0) + " ms");

}

}

Hasil Perbandingan Waktu Eksekusi

Queue (Tambah 1000 pelanggan): 3.38775 ms

Stack (Tambah 1000 histori): 0.637667 ms

Linked List (Tambah 1000 mahasiswa): 3.32525 ms

BUILD SUCCESSFUL (total time: 0 seconds)

Catatan: Waktu bisa bervariasi tergantung spesifikasi mesin dan kondisi runtime.\*

**5. Perbandingan Antara Queue, Stack, dan Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fitur** | **Queue** | **Stack** | **Linked List** |
| Akses Data | FIFO | LIFO | Akses bebas via iterasi |
| Operasi Utama | add, remove | push, pop | add, remove, get |
| Kompleksitas | O(1) untuk operasi dasar | O(1) untuk push/pop | O(1) insert/delete di awal/akhir |
| Fleksibilitas | Terbatas pada FIFO | Terbatas pada LIFO | Sangat fleksibel untuk dinamika |

**6. Kesimpulan: Kelebihan & Kekurangan**

Queue

- Kelebihan: Cocok untuk antrian proses, buffer data, dll.

- Kekurangan: Tidak fleksibel untuk akses elemen secara acak.

Stack

- Kelebihan: Sederhana dan cepat untuk operasi LIFO, cocok untuk rekursi dan backtracking.

- Kekurangan: Tidak cocok untuk pengolahan data yang perlu diakses secara acak.

Linked List

- Kelebihan: Dinamis, tidak perlu alokasi memori awal.

- Kekurangan: Akses data lebih lambat dibanding array karena harus traversal.