LAPORAN PRAKTIKUM

STRUKTUR DATA



DISUSUN OLEH :

Sasya Zamora

2411533014

DOSEN PENGAMPU :

Dr. Wahyudi, S.T.M.T

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

2024

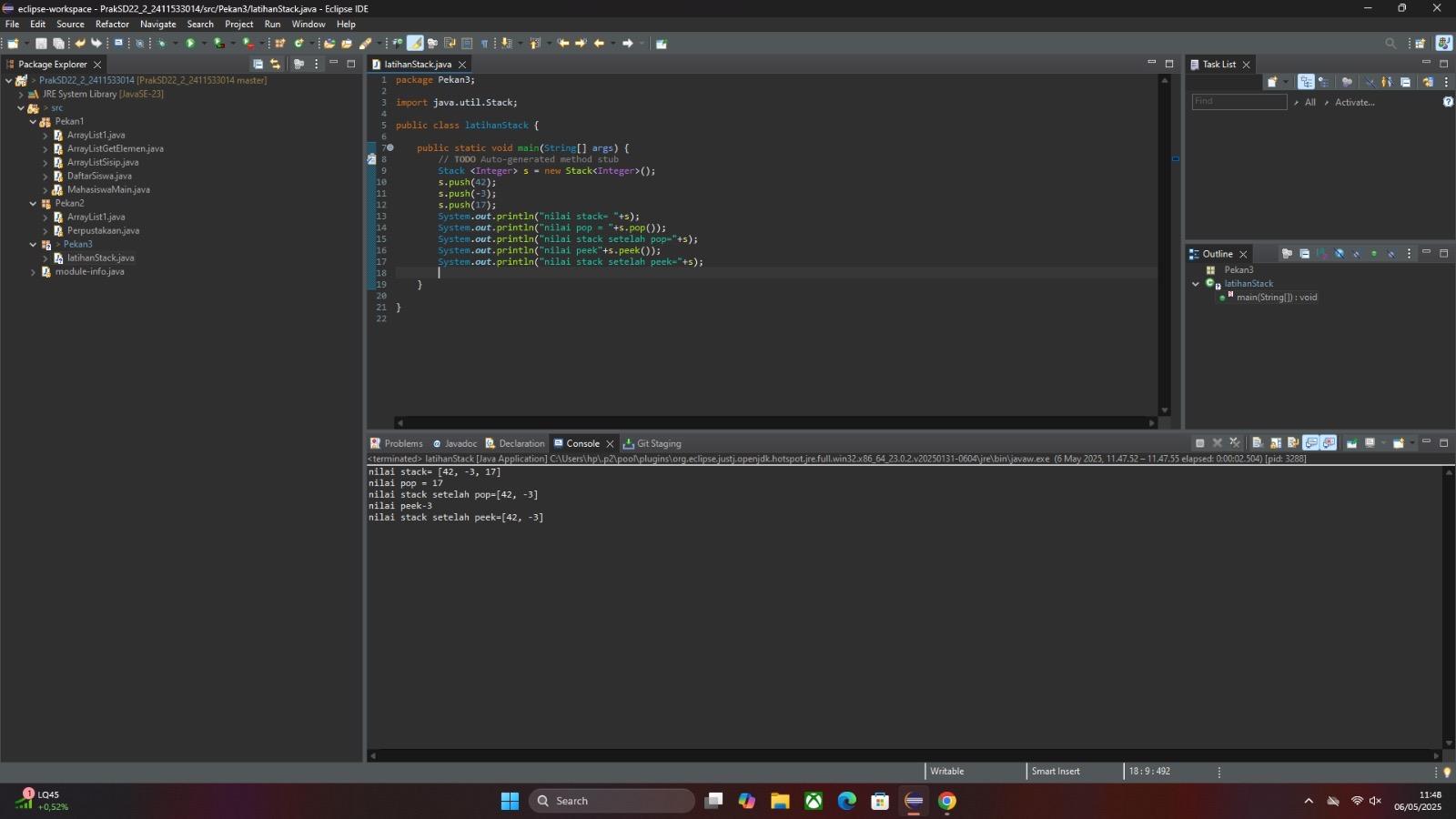
1. TUJUAN
2. Memahami konsep Stack.
3. Mempelajari operasi dasar Stack (push, pop, peek, isEmpty, isFull).
4. Menerapkan konsep Stack dalam program.
5. PEMBAHASAN

Pratikum ini berfokus pada pemahaman mendalam tentang struktur data Stack (tumpukan), sebuah struktur data LIFO (Last-In, First-Out). Mahasiswa akan mempelajari implementasi praktis dari operasi-operasi fundamental Stack, yaitu push (menambahkan elemen ke puncak tumpukan), pop (menghapus elemen dari puncak tumpukan), peek (melihat elemen teratas tanpa menghapusnya), isEmpty (mengecek apakah tumpukan kosong), dan isFull (mengecek apakah tumpukan penuh). Pemahaman ini akan diuji melalui implementasi program yang melibatkan penggunaan Stack.

Lebih lanjut, pratikum ini menuntut mahasiswa untuk menganalisis kompleksitas algoritma yang terkait dengan operasi-operasi Stack. Analisis ini penting untuk memahami efisiensi dan performa dari implementasi Stack yang berbeda. Mahasiswa diharapkan mampu membandingkan berbagai pendekatan implementasi dan memilih pendekatan yang paling tepat berdasarkan kebutuhan spesifik suatu permasalahan.

Akhirnya, pratikum ini bertujuan untuk memperluas wawasan mahasiswa tentang aplikasi Stack dalam berbagai konteks. Stack memiliki peran penting dalam berbagai algoritma dan struktur data lainnya, serta digunakan dalam berbagai aplikasi pemrograman seperti manajemen memori, pemrosesan ekspresi aritmatika, dan pemanggilan fungsi (function call stack). Dengan memahami aplikasi-aplikasi ini, mahasiswa akan mampu menghubungkan konsep teoritis dengan penerapan praktisnya di dunia nyata.

1. LANGKAH PRAKTIKUM
2. LatihanStack



1. Import Statement:

import java.util.Stack;

Baris ini mengimpor kelas Stack dari library java.util. Stack adalah implementasi dari struktur data tumpukan (stack) dalam Java. Kita perlu mengimpornya agar kita bisa menggunakannya dalam kode kita.

2. Class Definition:

public class latihanStack {

Ini mendefinisikan sebuah kelas bernama latihanStack. Kata kunci public berarti kelas ini dapat diakses dari mana saja.

3. Main Method:

public static void main(String[] args) {

Ini adalah method main, titik awal eksekusi program Java. Semua kode yang ingin kita jalankan akan berada di dalam kurung kurawal {} dari method main.

4. Stack Initialization:

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

Sebuah objek Stack bernama s di-instansiasi. <Integer> menspesifikasikan tipe data yang akan disimpan dalam stack, yaitu bilangan bulat (Integer). new Stack<Integer>() menciptakan objek Stack yang kosong.

5. Push Operations:

s.push(42);

s.push(-3);

s.push(17);

Tiga baris kode ini menambahkan tiga bilangan bulat (42, -3, dan 17) ke dalam stack s menggunakan method push(). push() menambahkan elemen ke atas stack. Ingat bahwa stack mengikuti prinsip LIFO (Last-In, First-Out).

6. Printing the Stack:

System.out.println("nilai stack - " + s);

Baris ini mencetak isi stack s ke konsol. Output akan berupa [42, -3, 17]. Perhatikan bahwa elemen 17 berada di atas karena merupakan elemen terakhir yang ditambahkan.

7. Pop Operation:

System.out.println("nilai pop - " + s.pop());

s.pop() menghapus dan mengembalikan elemen teratas dari stack. Dalam hal ini, 17 akan dihapus, dan nilai 17 akan dicetak ke konsol.

8. Printing the Stack After Pop:

System.out.println("nilai stack setelah pop - " + s);

Setelah pop(), stack s akan berisi [42, -3]. Baris ini mencetak isi stack yang telah diperbarui.

9. Peek Operation:

System.out.println("nilai peek - " + s.peek());

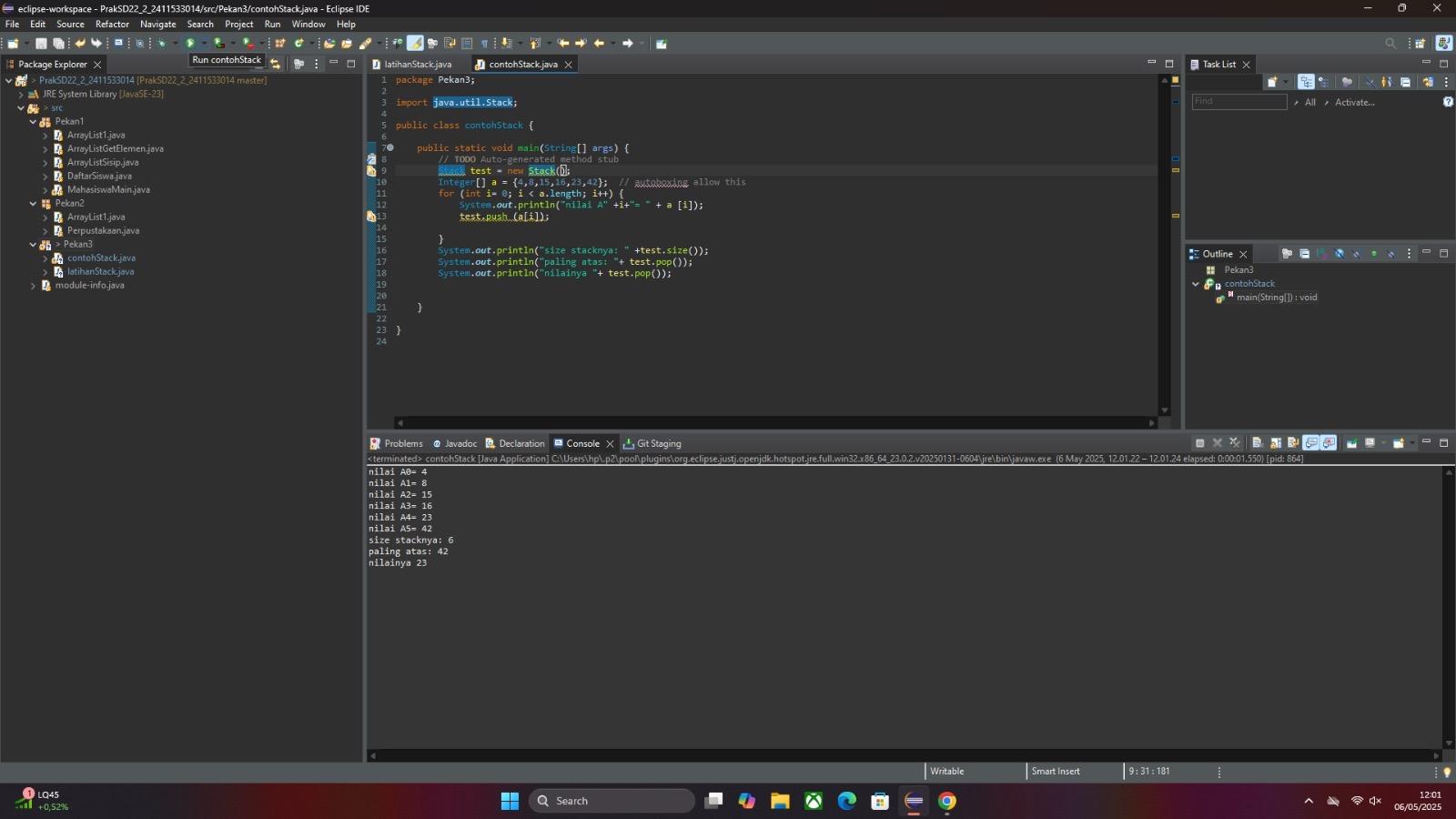
s.peek() mengembalikan elemen teratas dari stack tanpa menghapusnya. Output akan berupa 42.

10. Printing the Stack After Peek:

System.out.println("nilai stack setelah peek - " + s);

Stack s tetap berisi [42, -3] karena peek() hanya melihat elemen teratas, tidak menghapusnya.

1. ContohStack



1. Import Statement:

import java.util.Stack;

Mengimpor kelas Stack yang dibutuhkan dari library java.util.

2. Class Declaration:

public class contohStack {

Mendefinisikan kelas bernama contohStack.

3. Main Method:

public static void main(String[] args) {

Method main, titik awal eksekusi program.

4. Stack Initialization and Population:

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

int[] A = {4, 8, 15, 16, 23}; // Array of integers

for (int i = 0; i < A.length; i++) {

test.push(A[i]);

}

Sebuah stack bernama test diinisialisasi untuk menyimpan bilangan bulat. Kemudian, sebuah loop for menambahkan elemen-elemen dari array A ke dalam stack test menggunakan method push(). Elemen ditambahkan satu per satu ke atas stack.

5. Printing Values:

System.out.println("nilai A0 - " + A[0]);

System.out.println("nilai A1 - " + A[1]);

System.out.println("nilai A2 - " + A[2]);

System.out.println("nilai A3 - " + A[3]);

System.out.println("nilai A4 - " + A[4]);

Menampilkan nilai-nilai dari array A ke konsol. Ini bukan bagian dari operasi stack itu sendiri, tetapi memberikan konteks tentang nilai-nilai yang telah ditambahkan ke stack.

6. Getting Stack Size and Top Element:

System.out.println("size stacknya: " + test.size());

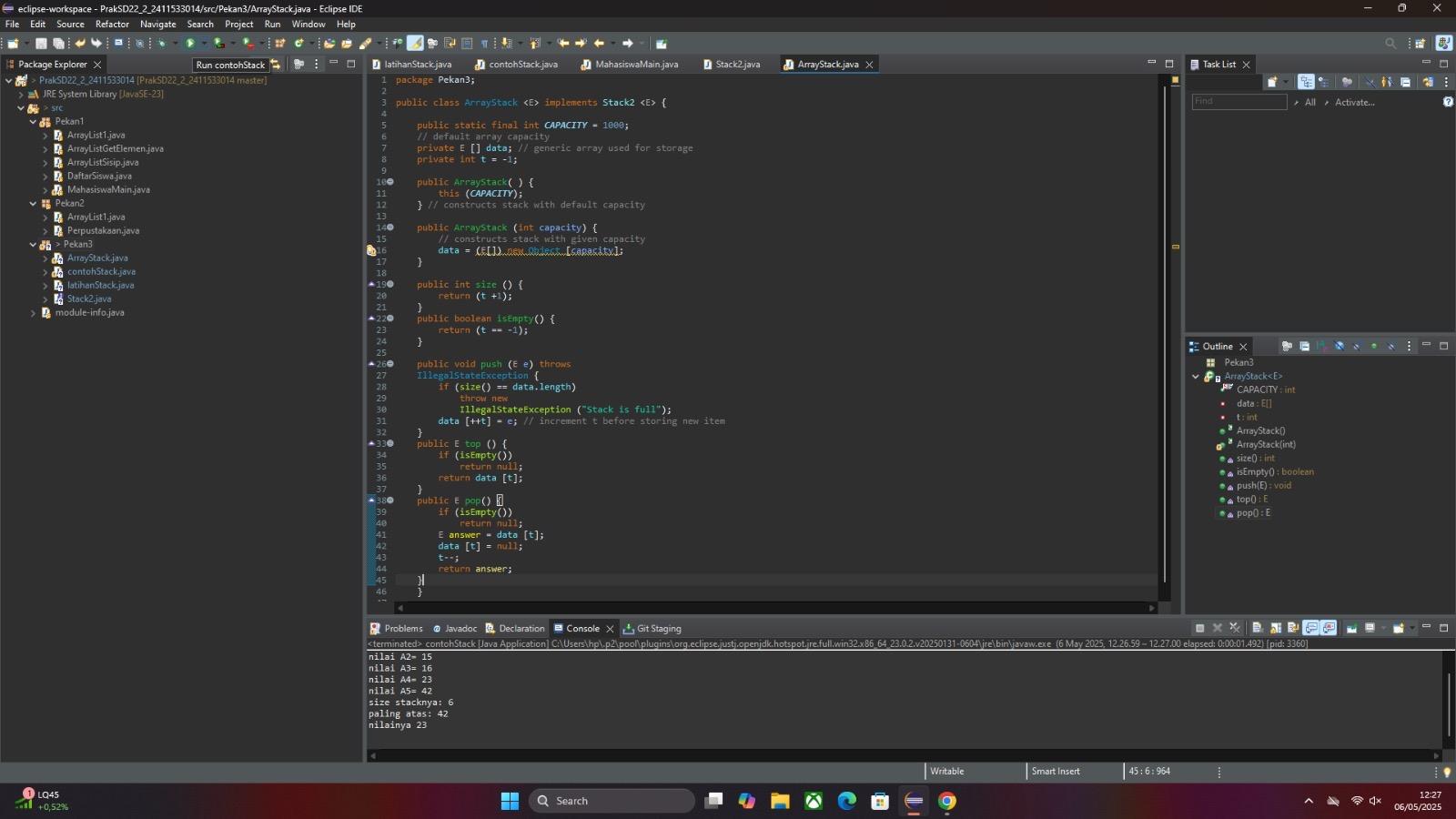
System.out.println("paling atas: " + test.peek());

System.out.println("nilainya " + test.pop());

• test.size(): Mengembalikan jumlah elemen dalam stack (test).

• test.peek(): Mengembalikan elemen teratas stack tanpa menghapusnya.

• test.pop(): Menghapus dan mengembalikan elemen teratas stack.

1. ArrayStack

1. Deklarasi Kelas dan Antarmuka:

public class ArrayStack<E> implements Stack<E> {

Ini mendeklarasikan kelas ArrayStack yang bersifat generik (<E>), artinya dapat menyimpan berbagai tipe data. Implementasi Stack<E> menunjukkan bahwa kelas ini mengimplementasikan antarmuka Stack (yang kemungkinan besar didefinisikan di tempat lain atau diimpor dari suatu library). Antarmuka ini mendefinisikan kontrak untuk operasi-operasi stack dasar.

2. Konstanta Kapasitas:

public static final int CAPACITY = 1000;

Mendefinisikan konstanta CAPACITY dengan nilai 1000. Ini menentukan ukuran maksimum array yang digunakan untuk menyimpan elemen stack.

3. Variabel Instansi:

private int size = 0; // Number of elements in the stack

private E[] data; // Generic array used for storage

• size: Variabel integer untuk melacak jumlah elemen yang ada di stack.

• data: Array generik (E[]) yang digunakan untuk menyimpan elemen stack. Perhatikan bahwa array ini diinisialisasi nanti.

4. Konstruktor:

public ArrayStack() { this(CAPACITY); } // Constructs stack with default capacity

public ArrayStack(int capacity) { // Constructs stack with given capacity

data = (E[]) new Object[capacity]; // Safe cast; compiler may give warning.

}

Ada dua konstruktor:

\* Konstruktor pertama membuat stack dengan kapasitas default (1000).

\* Konstruktor kedua memungkinkan untuk menentukan kapasitas stack saat diinisialisasi. Perhatikan penggunaan casting (E[]) new Object[capacity] yang diperlukan karena kita tidak dapat membuat array generik secara langsung di Java.

5. Method-method Utama:

public int size() { return size; }

public boolean isEmpty() { return size == 0; }

public void push(E e) throws IllegalStateException {

if (size == data.length) throw new IllegalStateException("Stack is full");

data[size++] = e; // Increment before storing new item

}

public E top() {

if (isEmpty()) return null;

return data[size - 1];

}

public E pop() {

if (isEmpty()) return null;

E answer = data[--size]; // Decrement before returning

data[size] = null; // Help with garbage collection

return answer;

}

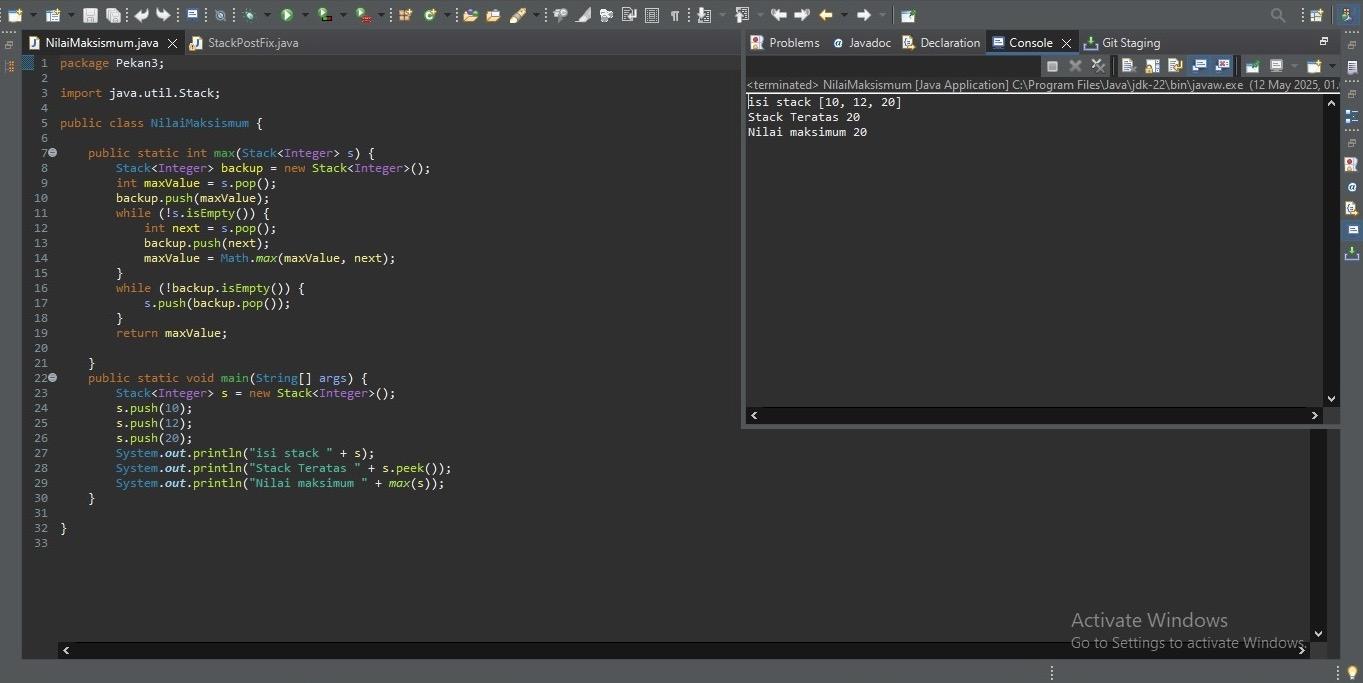
• size(): Mengembalikan jumlah elemen di stack.

• isEmpty(): Mengembalikan true jika stack kosong, false jika tidak.

• push(E e): Menambahkan elemen e ke atas stack. Memunculkan IllegalStateException jika stack penuh.

• top(): Mengembalikan elemen teratas stack tanpa menghapusnya. Mengembalikan null jika stack kosong.

• pop(): Menghapus dan mengembalikan elemen teratas stack. Mengembalikan null jika stack kosong.

1. NilaiMaksimum

Fungsi max:

public static int max(Stack<Integer> s) {

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

int maxValue = s.pop(); // Ambil elemen teratas

backup.push(maxValue); // Simpan ke dalam backup stack

while (!s.isEmpty()) {

int next = s.pop(); // Ambil elemen berikutnya

maxValue = Math.max(maxValue, next); // Bandingkan dengan maxValue

backup.push(next); // Simpan ke dalam backup stack

}

while (!backup.isEmpty()) {

s.push(backup.pop()); // Kembalikan elemen ke stack asli

}

return maxValue;

}

Fungsi ini menerima sebuah stack integer (s) sebagai input dan mengembalikan nilai maksimum di dalam stack tersebut. Berikut langkah-langkahnya:

1. Inisialisasi: Membuat stack baru bernama backup untuk menyimpan elemen-elemen sementara. Elemen teratas stack asli (s) diambil dan disimpan di maxValue dan juga di backup.

2. Iterasi: Loop while pertama berjalan selama stack s tidak kosong. Pada setiap iterasi, elemen teratas s diambil (next), dibandingkan dengan maxValue menggunakan Math.max(), dan yang terbesar disimpan di maxValue. next juga disimpan di backup.

3. Pemulihan: Loop while kedua mengembalikan semua elemen dari backup ke stack asli (s) dalam urutan yang benar (karena backup menggunakan LIFO seperti stack asli).

4. Pengembalian Nilai: Fungsi mengembalikan maxValue, yang berisi nilai maksimum yang ditemukan.

Fungsi main:

public static void main(String[] args) {

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

s.push(10);

s.push(12);

s.push(20);

System.out.println("isi stack " + s);

System.out.println("Stack Teratas " + s.peek());

System.out.println("Nilai maksimum " + max(s));

}

Fungsi main melakukan hal berikut:

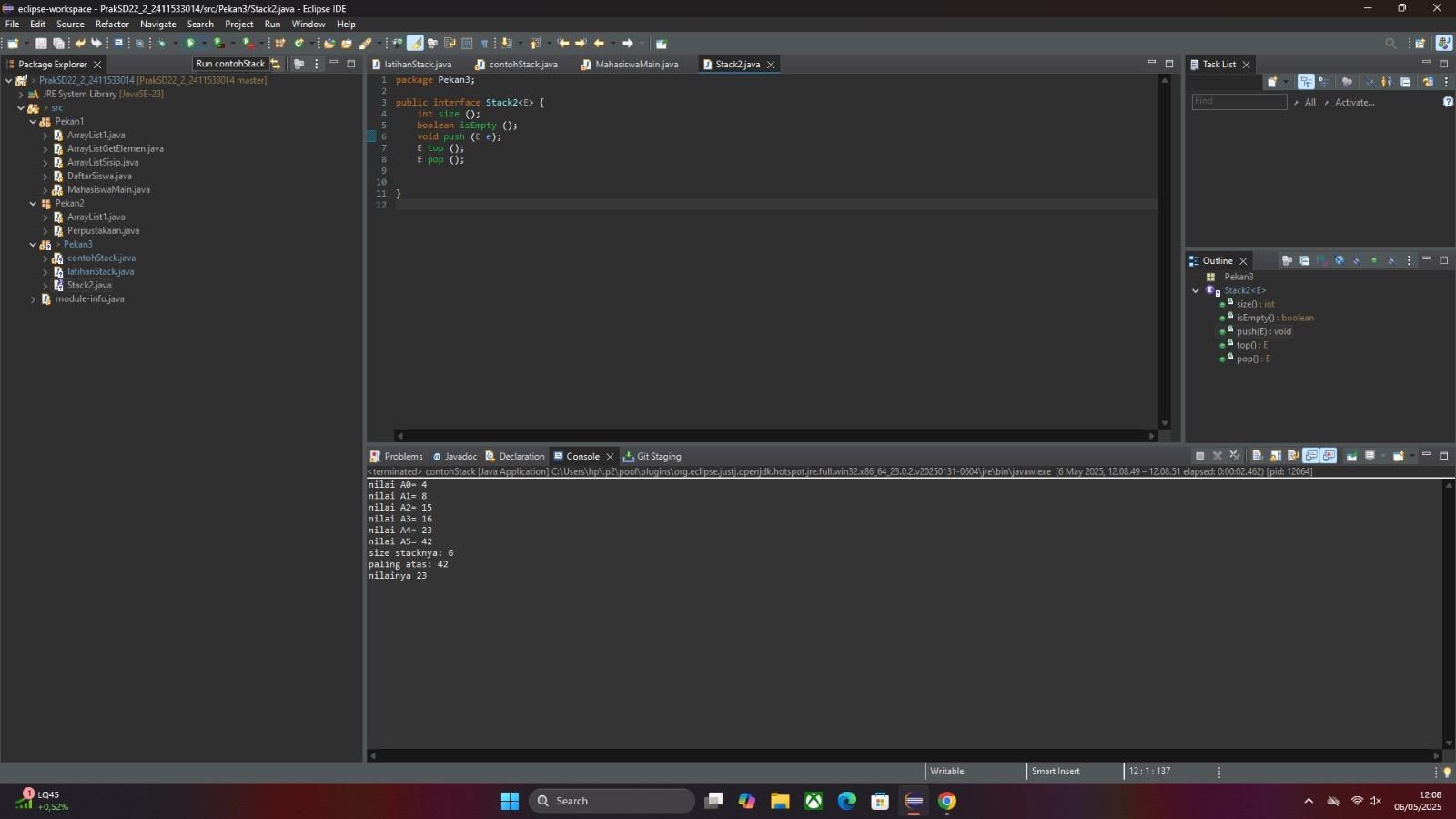
1. Inisialisasi: Membuat stack integer baru (s).

2. Penambahan Elemen: Menambahkan tiga elemen (10, 12, 20) ke stack s.

3. Cetak Stack: Mencetak isi stack s.

4. Cetak Elemen Teratas: Mencetak elemen teratas stack s menggunakan s.peek().

5. Panggil Fungsi max: Memanggil fungsi max untuk menemukan nilai maksimum dalam stack s dan mencetak hasilnya.

1. Stack2 

package Pekan3;

public interface Stack2<E> {

​int size ();

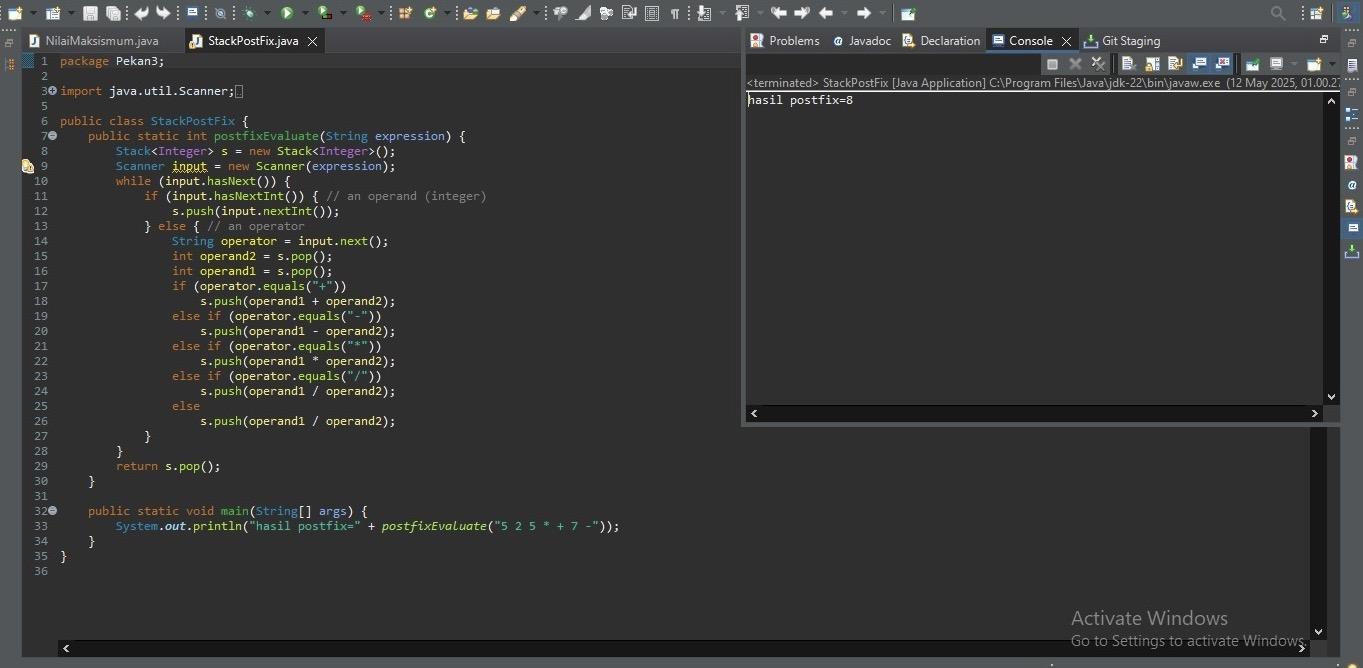
​boolean isEmpty ();

​void push (E e);

​E top ();

​E pop ();

}

1. StackPostfix

public static int postfixEvaluate(String expression) {

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

Scanner input = new Scanner(expression);

while (input.hasNext()) {

if (input.hasNextInt()) { // Jika input adalah angka (operand)

s.push(input.nextInt());

} else { // Jika input adalah operator

String operator = input.next();

int operand2 = s.pop();

int operand1 = s.pop();

if (operator.equals("+")) {

s.push(operand1 + operand2);

} else if (operator.equals("-")) {

s.push(operand1 - operand2);

} else if (operator.equals("\*")) {

s.push(operand1 \* operand2);

} else if (operator.equals("/")) {

s.push(operand1 / operand2);

} else {

// Handle error: invalid operator

}

}

}

return s.pop(); // Hasil akhir ada di puncak stack

}

Method postfixEvaluate menerima string expression yang merepresentasikan ekspresi postfix sebagai input. Ia menggunakan stack untuk mengevaluasi ekspresi tersebut.

Langkah-langkah Eksekusi:

1. Inisialisasi: Sebuah stack integer (s) dan Scanner (input) untuk parsing expression dibuat.

2. Iterasi: Loop while memproses setiap token (angka atau operator) dalam expression.

3. Operand: Jika token adalah angka (diperiksa dengan input.hasNextInt()), angka tersebut di-push ke stack.

4. Operator: Jika token adalah operator (+, -, \*, /), dua operand di-pop dari stack (operand2 dan operand1), operasi yang sesuai dilakukan, dan hasilnya di-push kembali ke stack.

5. Penanganan Error (Implicit): Tidak ada penanganan error eksplisit untuk operator yang tidak valid. Jika operator yang tidak dikenal ditemukan, program akan mengalami error EmptyStackException karena s.pop() akan dipanggil pada stack yang kosong.

6. Hasil: Setelah semua token diproses, hasil akhir akan berada di puncak stack, dan nilai ini dikembalikan oleh method.

Fungsi main:

public static void main(String[] args) {

System.out.println("hasil postfix=" + postfixEvaluate("5 2 5 \* + 7 -"));

}

Fungsi main memanggil postfixEvaluate dengan ekspresi postfix "5 2 5 \* + 7 -". Ekspresi ini setara dengan (5 + (2 \* 5)) - 7 = 8.

1. ContohStack2

(Di ss dari Github karena dilaptop bermasalah pak 🙏🏻)

1. Deklarasi Package dan Import:

package Pekan3;

import java.util.Stack;

• package Pekan3;: Menempatkan kode dalam package Pekan3.

• import java.util.Stack;: Mengimpor kelas Stack dari library standar Java. Ini kemungkinan besar digunakan untuk perbandingan atau sebagai alternatif implementasi.

2. Kelas contohStack2:

public class contohStack2 {

Mendefinisikan kelas utama program.

3. Method main:

public static void main(String[] args) {

// TODO Auto-generated method stub

ArrayStack test = new ArrayStack();

Integer[] a = {4, 8, 15, 16, 23, 42}; // autoboxing allow this

for (int i = 0; i < a.length; i++) {

System.out.println("nilai A" + i + "=" + a[i]);

test.push(a[i]);

}

System.out.println("size stacknya: " + test.size());

System.out.println("paling atas: " + test.top());

System.out.println("nilainya " + test.pop());

}

Method main melakukan hal berikut:

• Inisialisasi: Membuat instance ArrayStack bernama test. Sebuah array integer a diinisialisasi.

• Loop dan Push: Loop for iterasi melalui array a, mencetak setiap elemen, dan menambahkannya ke stack test menggunakan method push().

• Informasi Stack: Setelah loop selesai, kode mencetak ukuran stack (test.size()), elemen teratas (test.top()), dan kemudian menghapus dan mencetak elemen teratas (test.pop()).

1. KESIMPULAN

Kode-kode Java yang telah kita bahas menunjukkan berbagai cara untuk mengimplementasikan dan menggunakan struktur data stack. Kita telah melihat contoh-contoh yang menggunakan kelas Stack bawaan Java, implementasi stack kustom menggunakan array, dan penggunaan interface untuk mendefinisikan kontrak untuk kelas stack. Contoh-contoh tersebut juga mencakup operasi dasar stack seperti push, pop, peek, size, dan isEmpty, serta aplikasi yang lebih kompleks seperti evaluasi ekspresi postfix dan pencarian nilai maksimum dalam stack. Penggunaan stack yang tepat sangat bergantung pada kebutuhan spesifik aplikasi dan pertimbangan efisiensi. Implementasi yang berbeda (menggunakan array atau linked list, misalnya) akan memiliki trade-off dalam hal performa dan penggunaan memori.