

LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 3



Oleh :

FADIL INSANUS SIDDIK

NIM 2411532013

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU : DR. WAHYUDI, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, Mei 2025

A. Pendahuluan

Struktur data merupakan salah satu konsep penting dalam ilmu komputer yang digunakan untuk menyimpan dan mengelola data secara efisien. Salah satu jenis struktur data yang sering digunakan adalah stack. Stack merupakan struktur data linear yang menggunakan prinsip Last In, First Out (LIFO), yaitu data yang terakhir dimasukkan akan menjadi data pertama yang dikeluarkan.

Dalam praktiknya, stack dapat digunakan pada berbagai kasus, seperti fitur undo/redo, navigasi halaman pada browser, serta dalam pemanggilan fungsi secara berurutan pada program. Operasi dasar dari stack meliputi push (menambahkan elemen), pop (menghapus elemen teratas), dan peek (melihat elemen teratas tanpa menghapusnya).

Melalui praktikum ini, mahasiswa diharapkan dapat memahami cara kerja stack, baik dari segi konsep maupun implementasinya menggunakan bahasa pemrograman Java.

B. Stack

1. public interface Stack2<E>

- Dibutuhkan untuk menjalankan beberapa program pada class-class selanjutnya

```
public interface Stack2<E> {
```

- Ini adalah interface generik bernama Stack2, dengan parameter E (untuk tipe data apa pun, misalnya Integer, String, dll).
- Interface adalah template kontrak yang harus diikuti oleh kelas yang mengimplementasikannya.

```
int size();
```

- Mengembalikan jumlah elemen dalam stack.

```
boolean isEmpty();
```

- Mengecek apakah stack kosong (true jika kosong).

```
void push(E e);
```

- Menambahkan elemen bertipe E ke atas stack.

```
E top();
```

- Mengembalikan elemen paling atas tanpa menghapusnya.

```
E pop();
```

- Menghapus dan mengembalikan elemen paling atas dari stack.

2. ArrayStack

```
2  
3 public class ArrayStack<E> implements Stack2<E>{
```

- Mendeklarasikan kelas ArrayStack dengan parameter generik E (untuk menyimpan tipe data apa saja).
- implements Stack2<E> artinya kelas ini harus memenuhi kontrak dari interface Stack2.
- Mendeklarasikan kelas ArrayStack dengan **parameter generik E** (untuk menyimpan tipe data apa saja).
- implements Stack2<E> artinya kelas ini harus memenuhi kontrak dari **interface Stack2**.

```
    public static final int CAPACITY = 1000;  
    private E[] data;  
    private int t = -1;
```

- CAPACITY: kapasitas default stack (1000 elemen).
- data: array generik untuk menyimpan elemen-elemen stack.
- t: indeks dari elemen teratas stack; dimulai dari -1 artinya stack kosong.

```
    public ArrayStack() {  
        this(CAPACITY);  
    }
```

- Konstruktor default. Jika objek dibuat tanpa parameter, maka kapasitasnya di-set ke 1000.

```
    public ArrayStack(int capacity) {  
        data = (E[]) new Object[capacity];  
    }
```

- Konstruktor dengan parameter kapasitas.

- Karena Java tidak mendukung langsung membuat array generik, maka dilakukan casting (E[]).

```
public int size() {  
    return (t + 1);  
}
```

- Mengembalikan jumlah elemen dalam stack.
- Karena t adalah indeks (mulai dari -1), maka jumlah elemen adalah t + 1.

```
public boolean isEmpty() {  
    return (t == -1);  
}
```

- Mengecek apakah stack kosong (jika t == -1).

```
public void push(E e) throws IllegalStateException{  
    if(size() == data.length)  
        throw new IllegalStateException("Stack is Full");  
    data[++t] = e;  
}
```

- Menambahkan elemen e ke atas stack.
- Dicek dulu apakah stack penuh. Jika ya, lempar exception.
- ++t: menaikkan indeks dulu, lalu masukkan elemen ke indeks tersebut.

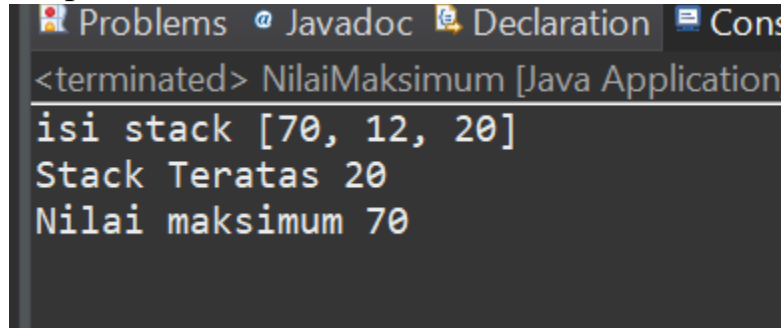
```
public E top() {  
    if (isEmpty())  
        return null;  
    return data[t];  
}
```

- Mengembalikan elemen paling atas dari stack tanpa menghapusnya.
- Jika stack kosong, kembalikan null.

```
public E pop() {  
    if (isEmpty())  
        return null;  
    E answer = data[t];  
    data[t] = null;  
    t--;  
    return answer;  
}
```

- Menghapus dan mengembalikan elemen paling atas.
- Jika kosong, return null.
- Simpan elemen atas ke variabel answer, lalu kosongkan posisinya (null), lalu turunkan t.

Output:



```
<terminated> NilaiMaksimum [Java Application]
isi stack [70, 12, 20]
Stack Teratas 20
Nilai maksimum 70
```

3. contohStack

```
import java.util.*;
```

- Mengimpor semua class dari package java.util, termasuk Stack, ArrayList, dll.

```
public class contohStack {
```

- Deklarasi kelas bernama contohStack.

```
public static void main(String[] args) {
```

- Metode utama (main) Java, titik masuk saat program dijalankan.

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

- Membuat objek stack bernama test untuk menyimpan bilangan bulat (Integer).
- Menggunakan kelas Stack dari Java yang bersifat LIFO (Last In First Out).

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

- Mendeklarasikan dan menginisialisasi array a bertipe Integer dengan 6 elemen.

```
Integer[] a = {4, 8, 15, 16, 23, 42};
```

- Perulangan dari indeks 0 hingga panjang array a untuk mengakses setiap elemen.

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

- Menampilkan elemen array a pada indeks i.

```
test.push(a[i]);
```

- Menambahkan elemen a[i] ke stack test.

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

- Menampilkan jumlah elemen dalam stack setelah semua data dimasukkan.

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

- Menampilkan elemen teratas dari stack tanpa menghapusnya menggunakan peek().

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

- Menghapus dan menampilkan elemen teratas dari stack menggunakan pop().

Output:

```
Problems @ Javadoc Declarati  
<terminated> contohStack [Java App  
nilai A0= 4  
nilai A1= 8  
nilai A2= 15  
nilai A3= 16  
nilai A4= 23  
nilai A5= 42  
size stacknya: 6  
paling atas: 42  
nilainya 42
```

4. latihanstack

```
import java.util.*;
```

- Mengimpor seluruh isi dari package java.util, termasuk kelas Stack.

```
public class latihanStack {
```

- Mendeklarasikan kelas Java bernama latihanStack.

```
    public static void main(String[] args) {
```

- Fungsi main() adalah titik awal eksekusi program Java.

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

- Membuat objek Stack bernama s untuk menyimpan nilai bertipe Integer.

```
        s.push(42);  
        s.push(-3);  
        s.push(17);
```

- Menambahkan elemen ke stack:
 - o 42 ditambahkan → stack = [42]
 - o -3 ditambahkan → stack = [42, -3]
 - o 17 ditambahkan → stack = [42, -3, 17]
- Urutan LIFO (Last In First Out), jadi 17 berada di paling atas.

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

- Menampilkan seluruh isi stack.
- Output akan seperti [42, -3, 17].

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

- Menghapus dan menampilkan elemen teratas stack (17).
- Stack setelah ini menjadi [42, -3].

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

- Menampilkan isi stack setelah pop().
- Output: [42, -3].

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

- Menampilkan elemen teratas stack tanpa menghapusnya.
- peek() akan menampilkan -3.

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

- Menampilkan isi stack lagi. Karena peek() tidak menghapus elemen, maka stack tetap [42, -3].

Output:

```
Problems @ Javadoc Declaration Console × Cov
<terminated> latihanStack [Java Application] C:\Users\LENOV
nilai stack = [42, -3, 17]
nilai pop = 17
nilai stack setelah pop = [42, -3]
nilai peek=3
nilai stack setelah peek= [42, -3]
```

5. NilaiMaksimum

```
import java.util.*;
```

- Mengimpor semua class dari java.util, termasuk Stack.

```
public class NilaiMaksimum {
```

- Mendeklarasikan class Java bernama NilaiMaksimum.

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

- Mendeklarasikan fungsi statis max yang menerima sebuah stack dan mengembalikan nilai maksimum dari stack tersebut.

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

- Membuat stack backup untuk menyimpan sementara data agar isi stack asli tidak hilang.

```
int maxValue = s.pop();
backup.push(maxValue);
```

- Mengambil elemen pertama dari stack (dengan pop) dan asumsikan itu nilai maksimum sementara.
- Simpan elemen itu ke dalam stack backup.


```
while (!s.isEmpty()) {
    int next = s.pop();
    backup.push(next);
    maxValue = Math.max(maxValue, next);
}
```

- Selama stack s belum kosong:
 - o Ambil elemen berikutnya (pop()),
 - o Simpan ke backup,
 - o Bandingkan dan perbarui nilai maksimum jika perlu.

```
while (!backup.isEmpty()) {
    s.push(backup.pop());
}
```

- Kembalikan semua elemen dari backup ke s agar isi stack tetap seperti semula.

```
return maxValue;
```

- Mengembalikan nilai maksimu yang ditemukan.

```
public static void main(String[] args) {
```

- Fungsi utama tempat program dijalankan.

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

- Membuat stack s bertipe Integer.

```
s.push(70);
s.push(12);
s.push(20);
```

- Menambahkan tiga angka ke dalam stack.
- Stack sekarang: [70, 12, 20] → 20 adalah elemen teratas.

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

- Menampilkan semua isi stack.

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

- Menampilkan elemen paling atas (tanpa menghapusnya).

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

- Memanggil fungsi max() dan menampilkan nilai maksimum dari elemen-elemen dalam stack.

Output:

```
<terminated> NilaiMaksimum [Java A  
isi stack [70, 12, 20]  
Stack Teratas 20  
Nilai maksimum 70
```

6. StackPostfix

```
import java.util.*;
```

- Mengimpor semua kelas dari java.util, termasuk Stack dan Scanner.

```
public class StackPostfix {
```

- Mendeklarasikan kelas utama dengan nama StackPostfix.

```
public static int postfixEvaluate(String expression) {
```

- Fungsi statis untuk mengevaluasi ekspresi postfix bertipe String, dan mengembalikan hasilnya dalam bentuk int.

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

- Membuat stack s untuk menyimpan angka selama proses evaluasi.

```
Scanner input = new Scanner(expression);
```

- Membuat objek Scanner untuk membaca ekspresi string kata per kata (dipisah spasi).

```
while (input.hasNext()) {
```

- Selama masih ada token (angka/operator) yang bisa dibaca dari ekspresi.

```
if (input.hasNextInt()) {  
    s.push(input.nextInt());  
}
```

- Jika token berikutnya adalah angka (integer), langsung dorong ke stack.

```
} else {  
    String operator = input.next();  
    int operand2 = s.pop();  
    int operand1 = s.pop();
```

- Membaca operator dan mengambil 2 angka terakhir dari stack (ingat: LIFO).
- operand1 adalah nilai sebelum operand2, karena di-stack urutannya dibalik.

```
if (operator.equals("+")){
    s.push(operand1 + operand2);
} else if (operator.equals("-")) {
    s.push(operand1 - operand2);
} else if (operator.equals("*")) {
    s.push(operand1 * operand2);
} else {
    s.push(operand1 / operand2);
}
```

- Mengecek jenis operator:
 - o (+, -, *, /)
 - o Setelah operasi dilakukan, hasilnya dikembalikan ke stack.

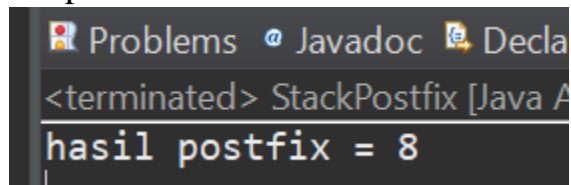
```
return s.pop();
```

- Setelah semua selesai, hasil akhir ada di atas stack → pop() untuk ambil nilainya.

```
public static void main(String[] args) {
    System.out.println("hasil postfix = "+postfixEvaluate("5 2 5 * + 7 -"));
}
```

- Fungsi utama untuk menjalankan program.
- Mengevaluasi ekspresi postfix "5 2 5 * + 7 -".
- Menampilkan hasilnya.

Output:



The screenshot shows an IDE interface with tabs for 'Problems', 'Javadoc', and 'Declarations'. Below the tabs, the text '<terminated> StackPostfix [Java A' is visible. The main output area displays 'hasil postfix = 8'.