LAPORAN PRAKTIKUM STRUKTUR DATA

Pekan 3

OLEH:

SEPTIAN RIYANDA PUTRA

(2411532016)

DOSEN PENGAMPU:

Dr. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. PENDAHULUAN

Struktur data Stack adalah salah satu bentuk struktur data linear yang bekerja dengan prinsip LIFO (Last In First Out), yaitu elemen yang terakhir dimasukkan akan menjadi elemen pertama yang dikeluarkan. Dalam bahasa pemrograman Java, Stack dapat diimplementasikan secara manual menggunakan array, atau dengan memanfaatkan class bawaan Java yaitu Stack. Tujuan dari praktikum ini adalah untuk mempelajari serta mengaplikasikan konsep Stack melalui berbagai bentuk implementasi, seperti stack berbasis array, evaluasi ekspresi postfix, dan pencarian elemen maksimum dalam stack.

B. TUJUAN

- Menerapkan konsep dasar struktur data Stack (LIFO) di Java.
- Membandingkan implementasi Stack menggunakan array dan class `Stack`.
- Melakukan analisis ekspresi postfix menggunakan `Stack`.
- Memahami proses pengembalian data yang tetap menjaga isi stack asli.

C. PROGRAM YANG DIBUAT

1. Class stack2.java

File ini berisi *interface* stack2 yang menjadi *blueprint* untuk stack generik berbasis array.

```
package pekan3;

public interface stack2 <E>{
    int size();
    boolean isEmpty();
    void  push (E e);
    E top();
    E pop();
}
```

Syntax `stack<E>` adalah generic interface agar stack dapat digunakan untuk berbagai tipe data. Metode `push`, `pop`, `size`, dan `isEmpty` adalah operasi dasar stack. Metode `top` dibuat mirip dengan `peek` pada stack

2. Class ArrayStack.java

```
private E[] data;
private int t = -1;
```

- data: array yang menyimpan elemen stack
- t: pointer/top stack dimulai dari -1 (stack kosong)

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

- `push`: menambahkan elemen ke atas stack

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

- memeriksa elemen teratas dan mengembalikan nilainya tanpa mengubahnya, mirip dengan metode `peek`

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

- Menghapus elemen teratas dan mengembalikannya dengan memanfaatkan metode `pop`

3. Class contohStack.java

```
ArrayStack test = new ArrayStack();
Integer[] a = {4, 8, 15, 16, 23, 42};
for(int i = 0; i < a.length; i++) {
    System.out.println("nilai A"+i+"= "+ a[i]);
    test.push(a[i]);
}
```

- Memanggil objek `ArrayStack` yang sudah dibuat sebelumnya
- Menambahkan elemen ke stack secara berurutan.

```
System.out.println("size stacknya: " + test.size());
System.out.println("paling atas: " + test.top());
System.out.println("nilainya " + test.pop());
```

- .`size`(): menghitung jumlah elemen yang ada
- .`top`(): mengambil elemen teratas tanpa menghapusnya
- .`pop`(): menghapus element teratas
- `size`(), `top`(), dan `pop`() dicetak secara berurutan

4. **Class latihanStack.java**

```
Stack<Integer> s = new Stack<Integer>();
s.push(item:42);
s.push(-3);
s.push(item:17);
```

- Panggil objek `Stack` dari `java.util`, masukkan beberapa elemen ke stack dengan metode `push`.

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

- Mencetak seluruh isi stack yang belum dimanipulasi
- Mencetak metode `pop` dari stack
- Mencetak stack yang sudah dimanipulasi dengan metode `pop`
- Mencetak metode `peek` dari stack yang sudah dimanipulasi
- Mencetak stack yang sudah dimanipulasi dengan metode `peek` dan `pop`

5. **Class NilaiMaksimum.java**

```
Stack<Integer> backup = new Stack<Integer>();
int maxValue = s.pop();
backup.push(maxValue);
```

- Mememanggil method Stack dari java.util lalu mengambil elemen sambil menyimpan ke stack backup.

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

- Membandingkan nilai saat ini dengan maksimum sebelumnya jika stack tidak kosong.

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

- Mengembalikan isi stack ke keadaan semula.

6. Class StackPostfix.java

```
if (input.hasNextInt()) {
    s.push(input.nextInt()); // operand
} else {
    String operator = input.next(); // operator
    int operand2 = s.pop();
    int operand1 = s.pop();
```

- Parsing input: angka dimasukkan ke stack, operator diproses langsung.

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

- Memeriksa kondisi, jika memenuhi maka operasi dilakukan pada dua operand teratas.

D. KESIMPULAN

Dalam praktikum ini, mahasiswa mempelajari mekanisme dan penerapan struktur data Stack, baik dengan membuatnya secara manual menggunakan array (ArrayStack) maupun dengan memanfaatkan class Stack dari Java. Prinsip LIFO (Last In First Out) dipraktikkan melalui operasi dasar seperti push, pop, dan peek. Selain itu, mahasiswa juga dapat mengaplikasikan Stack pada kasus nyata, seperti evaluasi ekspresi postfix dan pencarian nilai maksimum tanpa memodifikasi isi stack yang asli.